# EFFICIENT MINING OF ASSOCIATION RULES IN LARGE DATABASE

by

PUNIT KUMAR MISHRA



DEPARTMENT OF INDUSTRIAL AND MANAGEMENT ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

March, 2001

# EFFICIENT MINING OF ASSOCIATION RULES IN LARGE DATABASE

A

Thesis Submitted

In Partial Fulfillment of the Requirements

For the Degree of

**MASTER OF TECHNOLOGY**

**By**

**PUNIT KUMAR MISHRA**

to the

**Department of Industrial and Management Engineering**

**Indian Institute of Technology Kanpur**

**March, 2001**

## Acknowledgement

I owe a special debt of gratitude to my thesis supervisor Dr. Veena Bansal, under whose able and dynamic guidance the present work is carried out. I am heartily thankful to her for the confidence she showed in me. Her sincere encouragement and co-operative and painstaking attitude lead me complete this work in time.

I am also thankful to Mr. H.K.Mishra (UG Office) for providing me the registration data of IITK students.

I would also like to cherish the company of all my friends for their continuous moral support. My special thanks to Sagar who helped me in coding the algorithms.

Last but not the least I wish to acknowledge the constant encouragement and blessings which my parents gave me.

Punit Kumar Mishra

# CERTIFICATE

This is to certify that the present dissertation entitled **"Efficient Mining of Association Rules in Large Database"** has been carried out by Mr. Punit Kumar Mishra (Roll No. 9911411) under my supervision and that this work has not been submitted elsewhere for any degree.

**Dr. Veena Bansal**
Lecturer,
Deptt. of Industrial and Management Engg.
**Indian Institute of Technology**
**Kanpur-208016**
**India.**

March 22, 2001

# Index


P.No

1 Introduction 1-7

1.1 How do people use data mining

1 2 Reasons for growing popularity of Data Mining

1.3 Data mining and Data warehousing

1.4 Data mining as a Part of Knowledge discovery process

1.5 Brief overview of the work

2 Literature Survey 8-32

2.1 Tasks solved by Data Mining

2.2 Requirements and challenges of Data Mining

2.3 An overview of Data mining Techniques

2.3.1 Classifying data Mining Techniques

2.3.2 Mining Different kinds of knowledge from database

2 4 Association Rule Mining

2.4.1 Basic Concepts

2.4.2 Algorithms for Finding Association Rules in Database

2.4.3 Apriori Algorithm

2.4.4 Dynamic Itemset Counting(DIC) Algorithm

2.4.5 TID-LIST Based Approach

2.4.6 Other Algorithms

2.4.7 Parallel and Distributed Mining of Association Rules

3. Implementation and Study of Existing ARM Algorithms 33-42

3.1 DIC Algorithm

3.2 Apriori Algorithm

3.3 Tid-list based algorithm

3.4 Limitation in our implementation

3.5 Synthetic data generation

4. Problems in Handling Large databases 43-48
4.1 Our Approach for Local Pruning
5 Our New Approach for ARM 49-56
5 1 Method
5.2 Analysis
6 Practical Application of ARM 57-60
6.1 Motivation
6.2 Registration Data of IITK Students
6.3 Results
7. Data Miners Available and their features 61-65
8. Conclusion 66
Bibliography 67-68
Appendix

# List Of figures and Charts


| | P_No |
|---|---|
| 1.1 Data mining as a part of knowledge discovery | 5 |
| 2.1 Apriori Algorithm | 18 |
| 2 2 Counting Itemsets in Apriori and DIC | 22 |
| 2 3 An itemset Lattice | 24 |
| 2.4 DIC Algorithm (a, b, c, d) | 25,26 |
| 3.1 A trie with four items | 34 |
| 3.2 Performance of DIC algorithm | 38 |
| 3.3 tid-list Based Algorithm (Intersection Strategy) | 39 |
| 3.4 Performance of tid-list based algorithm | 40 |
| 3 5 Synthetic Data Generated with 12 items(sample) | 42 |
| 4.1 Affect of partitioning database on the performance | 44 |
| 4.2 Count Distribution Algorithm | 45 |
| 5.1 Comparative size of data | 53 |
| 5.2 Compression Ratio obtained | 54 |
| 5.3 Performance of My_mine | 55 |
| 5.4 Comparison with Aprioi | 56 |

# Abstract

Data mining is a process of extraction of implicit, previously unknown and potentially useful information from data in databases. One of the major technologies in data mining involves the discovery of association rules. The task of discovering all frequent associations in very large databases is quite challenging. The search space is exponential in number of database attributes and with very large databases, the problem of I/O minimization becomes paramount.

In present work we have first introduced the concepts of Data Mining with special emphasis on Association Rule Mining (ARM) and studied the performance of some existing ARM algorithms. We also introduced the concepts of distributed ARM and proposed a method for local pruning to increase the efficiency of distributed mining. We next proposed a new method for ARM which is based on data format conversion. It shows considerable performance improvement over apriori. Finally to gain the insight of how the ARM techniques can be applied to real life data, we did the mining of registration data of IIT Kanpur students and found association rules hidden in the data.

# Chapter-1

# INTRODUCTION

We live in the Age of Information. The importance of collecting data that reflect our business or scientific activities to achieve competitive advantage is widely recognized now. Powerful systems for collecting data and managing it in large databases are in place in all large and mid-range companies. However, the challenge of turning this data into one's success lies in the extraction of knowledge about the system from the collected data.

Recently our capabilities of both generating and collecting data have been increasing rapidly. The wide spread use of bar codes for most commercial products, the computerization of many business and government transactions, and the advances in data collection tools have provided us with huge amounts of data. Millions of databases have been used in business management, government administration, scientific and engineering data management and many other applications. It is noted that the number of such databases keeps growing very rapidly because of the availability of powerful and affordable database systems. This explosive growth in data and databases has generated an urgent need for new techniques and tools that can intelligently and automatically transform the processed data into useful information and knowledge. Consequently *data mining* has become a research area with increasing importance. [9]

Data mining, which is also referred to as *knowledge discovery in databases*, means *a process of nontrivial extraction of implicit, previously unknown and potentially useful information* (such as knowledge rules, constraints, regularities) from data in databases [4].

Data mining is a new paradigm of intelligent data analysis aiming to derive advantageous knowledge from the data. The rapid development of techniques in recent years has made data mining a key approach toward data intensive decision support.

**Other definitions for Data Mining:**

Data Mining refers to the mining or discovery of new information in terms of patterns or rules from vast amount of data [15].

"Data Mining, or Knowledge Discovery in Databases (KDD) as it is also known, is the nontrivial extraction of implicit, previously unknown, and potentially useful information from data. This encompasses a number of different technical approaches, such as clustering, data summarization, learning classification rules, finding dependency net works, analyzing changes, and detecting anomalies." [William J Frawley, Gregory Piatetsky-Shapiro and Christopher J Matheus][16]

"Data mining consists of variety of techniques to identify nuggets of information or decision-making knowledge in bodies of data, and extracting these in such a way that they can be put to use in the areas such as decision support, prediction, forecasting and estimation. The data is often voluminous, but as it stands of low value as no direct use can be made of it; it is the hidden information in the data that is useful" [Clementine User Guide][16]

Modern computer data mining systems self learn from the previous history of the investigated system, formulating and testing hypotheses about the rules, which this system obeys. When concise and valuable knowledge about the system of interest had been discovered, it can and should be incorporated into some decision support system which helps the manager to make wise and informed business decisions.

## 1.1 How do people use data mining? [2]:

Data mining collects, stores, and organizes data for use in areas such as medicine, finance, intelligence, law enforcement, defense, logistic, education, and process control. Many applications use data mining for promotions, marketing, and sales. We can also use this technology for diagnostics and anomaly detection. Example of uses of data mining include

- A credit bureau decides on loans based on observations of people with similar buying patterns, income, and credit.
- A supermarket organizes its merchandise based on buying patterns and information about associations between products.
- A pharmaceutical company analyzes prescriptions to send promotional material to target customers.

- An intelligence agency reviews spending patterns and travel data to detect abnormal behavior by its employees.
- A physician analyzes X-ray images to detect abnormal patterns.
- An airline reservation system uses information about travel patterns and trends to maximize seat utilization.

Although these kinds of applications have been used for quite some time, they relied on statistical analysis by hand and have only recently begun employing data mining technologies to analyze data and make correlations and predictions.

## 1.2 Reasons for the growing popularity of Data Mining

### Growing Data Volume

The main reason for necessity of automated computer systems for intelligent data analysis is the enormous volume of existing and newly appearing data that require processing. The amount of data accumulated each day by various business, scientific, and governmental organizations around the world is daunting. According to information from GTE research center, only scientific organizations store each day about 1 TB (terabyte!) of new information. And it is well known that academic world is by far not the leading supplier of new data. It becomes impossible for human analysts to cope with such overwhelming amounts of data.

### Limitations of Human Analysis

Two other problems that surface when human analysts process data are the inadequacy of the human brain when searching for complex, multifactorial dependencies in data, and the lack of objectiveness in such an analysis. A human expert is always a hostage of the previous experience of investigating other systems. Sometimes this helps, sometimes this hurts, but it is almost impossible to get rid of this fact.

### Low Cost of Machine Learning

One additional benefit of using automated data mining systems is that this process has a much lower cost than hiring an army of highly trained (and paid) professional statisticians. While data mining does not eliminate human participation in solving the task completely, it significantly simplifies the job and allows an analyst who is not a professional in statistics and programming to manage the process of extracting knowledge from the data.

## 1.3 Data Mining and Data Warehousing:

The goal of data warehouse is to support decision making with data. Data Mining can be used in conjunction with a data warehouse to help with certain types of decisions. Data Mining can be applied to operational databases with individual transactions. Data Mining helps extracting meaningful new patterns that cannot be found necessarily by merely querying or processing data or metadata in the data warehouse. Data Mining applications should therefore be strongly considered early, during the design of the data warehouse. Also data Mining tools should be designed to facilitate their use in conjunction with data warehouses.

## 1.4 Data Mining as a part of Knowledge Discovery Process[15]:

Knowledge discovery in databases (KDD) encompasses more than data mining. These are seven steps or phases in KDD process.

1. Data Selection (selecting the proper data relevant to the analysis task)
2. Data Cleaning (remove noise or irrelevant data)
3. Data Integration (combining multiple data sources)
4. Data Transformation (aggregating etc. to consolidate form for mining)
5. Data Mining (intelligent methods to extract patterns etc.)
6. Pattern evaluation (identify patterns based on interestingness measures)
7. Knowledge Presentation (visualizing and representing knowledge to user)

Fig. 1.1 describes these steps. As an example consider a transaction database maintained by a specialty customer goods retailer. Suppose the client data includes a customer name, zip code, phone number, date of purchase, item_code, price, quantity, and total amount. A variety of new knowledge can be discovered by the KDD processing on the client database. During ***data selection,*** data about specific items or categories of items, or from stores in a specific region or area of country, may be selected. The ***data cleaning*** process then may collect invalid zip codes or eliminate records with incorrect phone prefixes. ***Integration*** or enrichment typically enhances the data with additional sources of information. For example, given the client names and phone numbers, the store may import other data about the age, income, and credit rating and append them to each record. Data transformation and encoding may be done

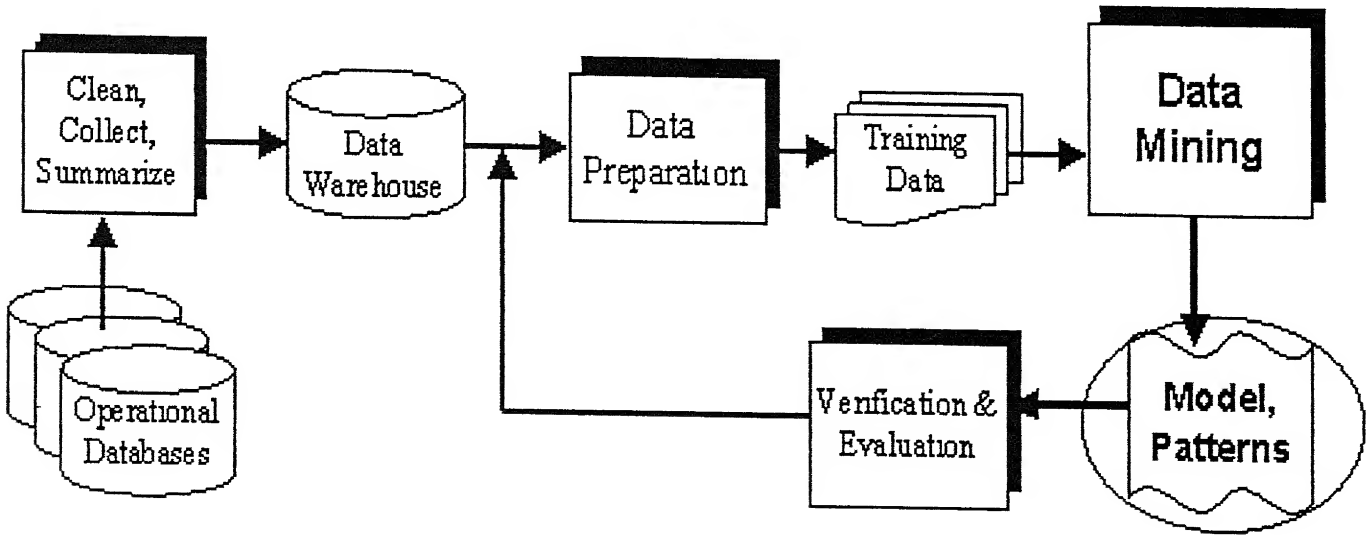


Fig 1.1

**Data Mining as a Part of Knowledge Discovery**

to reduce the amount of data. For instance, item codes may be grouped in terms of product categories into audio, video, supplies, electronic gadgets, camera accessories, and so on. It is only after such preprocessing that data Mining techniques are used to mine different rules and patterns. For example the result may be to discover:

- **Association rules**---e g., whenever a customer buys a video equipment, he or she also buys another electronic gadget.
- **Sequential patterns**---e.g., suppose a camera, and within three months he or she photographic supplies, and within six months an accessory item. A customer who buys more than twice in lean periods may be likely to buy at least once during Christmas period.
- **Classification trees**---e.g., customers may be classified by frequency of visits, by types of financing used, by amount of purchase, or by affinity for types of items, and some revealing statistics may be generated for such classes.

Data mining is not so much a single technique as the idea that there is more knowledge hidden in the data than shown itself on the surface. Form this point of view data mining is really an 'anything goes' affair. Any technique that helps extract more out of the data is useful, so data mining techniques form quite a heterogeneous group [14]

**Discovering association rules in Databases is the most appreciated and important data mining Problem**. Since its inception, association rule mining has become one of the core data mining tasks and has attracted tremendous interest among the researchers and practitioners. ARM (Association Rule Mining) is undirected or unsupervised data mining over variable length data and it produces clear, understandable results. It has an elegantly simple problem statement: to find the set of all subsets of items or attributes that frequently occur in many database records or transactions, and additionally, to extract rules on how a subset of items influences the presence of another subsets.

The prototype application of the ARM is *market-basket analysis*, where the items represent products, and the records represent point of sales data at large grocery stores. An example rule might be, " 90% of customers buying product A also buy product B." Other applications for ARM include customer segmentation, catalog design, store layout, and telecommunication alarm prediction.

**1.5 Brief overview of the work:** Chapter two covers literature survey. We have included basic concepts of general data mining and association rule mining in this chapter Then we have described the proposed algorithms for the association rule mining. We end this chapter with an introduction to parallel and distributed association rule mining

Chapter three covers the implementation aspects of these algorithms. The concentration was on three most widely used association rule mining algorithm. Then we have shown the limitations of the implementation and studied the performance of these algorithms in terms of the time taken by them to mine the association rules in the synthetic data which we generated for the analysis.

We discussed the problems in handling large databases in chapter four. We have proposed a method for pruning in distributed association rule mining without going into implementation details.

In chapter five we propose our new approach for association rule mining. We have shown the efficiency of our approach as compared to other algorithms

Chapter six covers a practical application of the association rule mining on the registration data of IIT Kanpur students.

In chapter seven some of the conclusions are presented.

# Chapter-2

# Literature Review

Data mining, *the extraction of hidden predictive information from large databases*, is a powerful new technology with great potential to help companies focus on the most important information in their data warehouses. Data mining tools predict future trends and behaviors, allowing businesses to make proactive, knowledge-driven decisions. The automated, prospective analyses offered by data mining move beyond the analyses of past events provided by retrospective tools typical of decision support systems. Data mining tools can answer business questions that traditionally were too time consuming to resolve. They scour databases for hidden patterns, finding predictive information that experts may miss because it lies outside their expectations.

## 2.1 Tasks Solved by Data Mining

**Predicting**

A task of learning a pattern from examples and using the developed model to predict future values of the target variable.

**Classification**

A task of finding a function that maps records into one of several discrete classes.

**Detection of relations**

A task of searching for the most influential independent variables for a selected target variable.

**Explicit modeling**

A task of finding explicit formulae describing dependencies between various variables.

**Clustering**

A task of identifying groups of records that are similar between themselves but different from the rest of the data. Often, the variables providing the best clustering should be identified as well.

**Market Basket Analysis**

Processing transactional data in order to find those groups of products that are sold together well. One also searches for directed association rules identifying the best product to be offered with a current selection of purchased products.

**Deviation Detection**

A task of determining the most significant changes in some key measures of data from previous or expected values.

## 2.2 Requirements and Challenges of Data Mining:

The data mining can be effective only if one clearly understands what kind of features an applied knowledge discovery system is expected to have and what kind of challenges one may face at the development stage of data mining techniques.[1]

1. ***Handling of different types of data:***

   Because there are many kinds of data and databases used in different applications, one may expect that a knowledge discovery system should be able to perform effective data mining on different kinds of data. Since most available databases are rational, it is crucial that a data mining system performs efficient and effective knowledge discovery on rational data. Moreover, many application databases contain complex data types, such as structured data and complex data objects, hypertext and multimedia data, spatial and temporal data, transaction data legacy data, etc. A powerful system should be able to perform effective data mining on such complex types of data as well. However, the diversity of data types and different goals of data mining to handle all kinds of data. Specific data mining on specific kinds of data, such as system dedicated to knowledge mining in relational databases, transaction database, spatial databases, multimedia databases, etc.

2. ***Efficiency and scalability of data mining algorithms:***

   To effectively extract information from a huge amount of data in databases, the knowledge discovery process must be efficient and scalable to large databases. That is, the running time of a data mining algorithm must be predictable and acceptable in large databases. Algorithms with exponential or even medium-order polynomial complexity will not be of practical use.

3. ***Usefulness, certainty, and expressiveness of data mining results:***

   The discovered knowledge should accurately portray the contents of the database and be useful for certain applications. The imperfectness should be expressed by measures of uncertainty, in the form of approximate rules or

quantitative rules. Noise and exceptional data should be handled elegantly in data mining systems. This also motivates a systematic study of measuring the quality of the discovered knowledge, including interestingness and reliability, by construction of statistical, analytical, and simulative models and tools.

*4.Expression of various kinds of data mining requests and results:*

Different kinds of knowledge can be discovered from a large amount of data. Also one may like to examine discovered knowledge from different views and present them in different forms. This requires us to express both the data mining requests and the discovered knowledge in high level languages or graphical user interfaces so that the data mining task can be specified by nonexperts and the discovered knowledge can be understandable and directly usable by users. This also requires the discovery system to adopt expressive knowledge representation techniques.

*5. Interactive mining knowledge at multiple abstraction levels:*

Since it is difficult to predict what exactly could be discovered from a database, a high-level data mining query should be treated as a probe, which may disclose some interesting traces for further exploration. Interactive discovery should be encouraged, which allows a user to interactively refine a data mining request, dynamically change data focusing, progressively deepen a data mining process, and flexibly view the data and data mining results at multiple abstraction levels from different angles.

*6. Mining information from different sources of data:*

The widely available local and wide-area computer network, including internet, connect many sources of data and form huge distributed, heterogeneous databases. Mining knowledge from different sources of formatted or unformatted data with diverse data semantics poses new challenges to data mining. On the other hand, data mining may help disclose the high level data regularities in heterogeneous databases, which can hardly be discovered by simple query systems. Moreover, the huge size of the database, the wide distribution of the data, and the computational complexity of some data mining methods motivate the development of parallel and distributed data mining algorithms.

*7. Protection of privacy and data security:*

When data can be viewed from many different angles and at different abstraction levels, it threatens the goal of protecting data security and guarding against the invasion of privacy. It is important to study when knowledge discovery may lead to invasion of privacy, and what security measures can be developed for preventing the disclosure of the sensitive information.

One can see that some of these requirements may carry conflicting goals. For example, the goal of protection of data security may conflict with the requirement of interactive mining of multiple level knowledge from different angles. Nevertheless we feel that it is still important to present an overall picture regarding the requirements of data mining.

## 2.3 An Overview of Data Mining Techniques:[4]

Since data mining poses many challenges research issues, direct application of methods and techniques developed in the related studies in machine learning, statistics, and database systems cannot solve these problems. It is necessary to perform dedicated studies to invent new data mining methods or develop integrated techniques for efficient and effective data mining. In this sense data mining itself has formed an independent new field.

### *2.3.1 Classifying Data Mining Techniques:*

There have been many advantages on researches and developments of data mining, and many data mining techniques and systems have recently been developed. Different classification schemes can be used to categorize data mining methods and systems based on the kinds of data bases to be studied, the kind of knowledge to be discovered, and the kinds of techniques to be utilized, as shown below.

- *What Kind of database to work on:*

  A data mining system can be classified according to the kinds of databases on which the data mining is performed. For example, a system is a relational data miner if it discovers knowledge from relational data or an object oriented one if it mines knowledge from object oriented databases. In general, a data miner can be classified according to its mining of knowledge from the following different

kinds of databases: relational databases, transaction databases, object oriented databases, deductive databases, spatial databases, temporal databases, multimedia databases, heterogeneous databases, active databases, legacy databases and the internet information-base.

- *What kind of knowledge to be Mined*

  Various kinds of knowledge can be discovered by data miners, including association rules, classification rules, discriminant rules, clustering, evolution and deviation analysis.

  Moreover, data miners can also be categorized according to the abstraction level of its discovered knowledge, which may be classified into generalized knowledge, primitive-level knowledge, and multiple level knowledge. A flexible data mining system may discover knowledge at multiple abstraction levels

- *What kinds of techniques to be utilized*

Data miners can also be categorized according to the underlying data mining techniques. For example, it can be categorized according to the driven method into autonomous knowledge miner, data driven miner, query-driven miner, and interactive data miner. It can also be categorized according to its underlying data mining approach into generalization based mining, pattern based mining, mining based on statistics or mathematical theories, and integrated approaches, etc.

### 2.3.2 Mining Different Kind of Knowledge from Databases

Data mining is application dependent and different applications may require different mining techniques. In general, the kinds of knowledge which can be discovered in a database can be categorized as follows.

Mining association rules in transactional or relational databases has recently attracted a lot of attention in data base communities. The task is to derive a set of strong association rules in the form of "$A_1$and .....and $A_m \Rightarrow B_1$ and .....and $B_n$" where $A_i$ ( for $i \in \{1,...,m\}$ ) and $B_j$ ( for $j \in \{1,....,n\}$ ) are set of attribute-values, from the data sets under consideration in a database. For example, one may find, from a large set of transaction data, an association rule such as "if a customer buys (one brand of) milk, he/she usually buys (another brand of) bread in the same transaction". Since mining association rule may require multiple scans through the large transaction database to

find different association patterns, the amount of processing can be huge. Some times the database may be larger in size than what fits in the core memory. In these cases the time spent on input/output operation can be considerable and thus performance improvement is an essential concern.

The most popularly used data mining and data analysis tools associated with database system products are data generalization and summarization tools, which carry several alternative names, such as on-line analytical processing (OLAP), multiple-dimensional databases, data cubes, data abstraction, generalization, summarization, characterization, etc Data generalization and summarization presents the general characteristics or a summarized high level view over a set of user specified data in database. For example, the general characteristic of the technical staff in a company can be described as a set of characteristic rules or a set of generalized summary tables. Moreover, it is often desirable to present generalized views about the data at multiple abstraction levels.

Another important application of data mining is the ability to perform classification in a huge amount of data. This is referred to as mining classification in a huge amount of data.

## 2.4 Association Rule Mining:

### 2.4.1 Basic Concepts:

An **association rule** is of the form

$$X \Rightarrow Y,$$

Where $X=\{x_1,x_2,\ldots,x_n\}$ and $Y=\{y_1,y_2,\ldots y_m\}$ are set of items, with $x_i$ and $y_j$ being distinct grocery items for all i and all j in a store (for market basket data). This association states that if a customer buys X, he or she is also likely to buy Y. In general, any association rule has the form LHS (left hand side) => RHS (right hand side), where LHS and RHS are the set of items. Association rules should supply both support and confidence.

The **support** for the rule LHS => RHS is the percentage of transaction that hold all of the items in the union, the set LHS $\cup$ RHS. If the support is low, it implies that there is not sufficient evidence that items in LHS $\cup$ RHS occur together, because the union happens in only a small fraction of transactions.

To compute confidence we consider the transactions that include items in LHS. The **confidence** for the association rule LHS => RHS is the percentage (fraction) of such transactions that also include RHS. Another term for confidence is the ***strength*** of the rule.

To clarify the concept of support and confidence let us take an example.

Consider four transactions in a random sample of the database: [15]

| Transaction-id | Items-Brought |
|---|---|
| 101 | Milk, bread, juice |
| 792 | Milk, juice |
| 1130 | Milk, eggs |
| 1735 | Bread cookies, coffee |

Number of transaction in which milk and juice appear together

n( milk and juice) = 2

Number of transactions in which bread and juice occur together

n(bread and juice) = 1

The rule milk => juice has 50% support. [ n (milk and juice) = 2]

The rule Bread => juice has only 25% support. [ n (bread and juice) = 1]

Confidence for the rule milk => juice is 66.7% (meaning that, of the three transactions in which milk occurs, two contain juice) and bread => juice has 50% confidence ( meaning that one of two transactions containing bread also contain juice).

Now we can formally define support and confidence as follows:

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of literals, called items. Let D be a set of transactions, where each transaction T is a set of items such that $T \subseteq I$. Each transaction is associated with an identifier , called TID. Let X be a set of items. A transaction T is said to contain X if and only if $X \subseteq T$. An association rule is an implication of the form $X \Rightarrow Y$, where $X \subset I$, $Y \subset I$ and $X \cap Y = \phi$. The rule $X \Rightarrow Y$ holds in the transaction set D with **confidence c** if c% of the transactions in D that contain X also contain Y. The rule $X \Rightarrow Y$ has **support s** in the transaction set D if s% of transactions in D contain $X \cup Y$.

### 2.4.2 Algorithms for finding Association rules in Databases:

Generating association rules involves two main steps [13]:

1. Find all frequent itemsets having a minimum support. The search space for enumeration of all frequent itemsets is $2^m$, which is exponential in m, the number of items. However, if we assume the transaction length has a bound, we can show that ARM is essentially linear in database size.
2. Generate strong rules having minimum confidence, from the frequent itemsets. For this we generate and test the confidence of all rules of the form X =>Y. Because we must consider each subset of X as a consequent, the rule generation step's complexity is $O(r.2^l)$, where r is the number of frequent itemsets, and l is the longest frequent itemset.

Many algorithms are proposed for association rule mining. The design space for these algorithms is composed of the following characteristics [13]:

***Bottom-up vs hybrid Search:***

One can easily see that an itemset is frequent iff all its subsets are also frequent. This property can be used to prune the search space for the frequent itemsets. There are three strategies proposed for pruning. The Association Rule Mining algorithms may differ in the manner they search the itemset lattice spanned by the subset relation. Many of the algorithms use the **bottom-up** search. In this strategy we start with individual items and then finding the their support we decide whether to consider a particular item in next iteration or not.

The top down approach starts with the top element of the lattice [17]. Its support is determined. If the support is greater than support threshold then there is no need to find the support of the subsets of this itemset. This prunes the search space considerably. The advantage of this approach is that if the maximal element is fairly large, then one can quickly identify it and one can avoid finding the support of all its subsets.

The hybrid scheme which combines the bottom-up and top down approach is based on the intuition that the greater the support of a frequent itemset the more likely it is to be a part of a longer frequent itemset.

*Complete vs. Heuristic Candidate Generation:*

ARM algorithms differ in the way they generate new candidates. A complete search, the dominant approach, guarantees that we can generate and test all frequent subsets. Here, complete doesn't mean exhaustive; we can use pruning to eliminate useless branches in the search space. Heuristic generation sacrifices the completeness for the sake of speed. At each step, it only examines a limited number of good branches. Random search to locate the maximal frequent itemsets is also possible. Methods that can be used here include genetic algorithms and simulated annealing. Because of a strong emphasis on completeness, ARM literature has not given much attention to the last two methods.

*Horizontal vs. Vertical Data Layout:*

In most of the cases the data is available in the horizontal layout. This is the reason, most of the algorithms use the horizontal data layout, which stores each customer's transaction id (**tid**) along with the items contained in the transaction. Some other algorithms use a vertical database layout, associating with each item its **tid-list**, which is a list of all transaction ids containing the item. Example for horizontal and vertical databases are shown below:

| Tid no | Item 1 | Item 2 | Item 3 | Item 4 | Item 5 | Item 6 |
|---|---|---|---|---|---|---|
| 1 | A | B | D | E | | |
| 2 | B | C | D | E | F | |
| 3 | A | D | F | | | |
| 4 | A | B | C | D | E | F |
| 5 | C | D | E | F | | |
| 6 | B | D | F | | | |
| 7 | A | C | E | F | | |
| 8 | D | E | | | | |
| 9 | A | C | D | E | F | |

**Horizontal Data Layout**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *A* | 1 | *3* | *4* | *7* | *9* | | | |
| *B* | *1* | *2* | *4* | *6* | | | | |
| *C* | *2* | *4* | *5* | *7* | *9* | | | |
| *D* | *1* | *2* | *3* | *4* | *5* | *6* | *8* | *9* |
| *E* | *1* | *2* | *3* | *5* | *7* | *8* | *9* | |
| *F* | *2* | *3* | *4* | *5* | *6* | *7* | *9* | |

**Vertical Data Layout**

### 2.4.3 Apriori Algorithm

The apriori algorithm by Rakesh Agrawal and colleagues has emerged as one of the basic ARM algorithms [3]. It serves as the base algorithm for most of the parallel algorithms Apriori uses a complete bottom-up search with a horizontal layout and enumerates all the frequent itemsets. An iterative algorithm, Apriori counts itemsets of a specific length in a given database pass. The process starts by scanning all transactions in the database and computing frequent 1-items. Next, a set of potentially frequent candidate 2- itemsets is formed from the frequent items. Another database scan finds their supports. The frequent two itemsets are retained for the next pass, and the process is repeated until all frequent itemsets have been enumerated.

The algorithm has three steps:

1. Generate candidates of length k from the frequent (k-1) length itemsets.
2. Prune any candidate that has at least one infrequent subset.
3. Scan all the transactions to obtain candidate supports. Apriori stores the candidates in a hash tree for fast support counting. In a hash tree, itemsets are stored in the leave.

At this point, one generates association rules that have a specified minimum accuracy too. Finally, we decide whether we really want those rules, in the light of your knowledge of the context. This is basically the APRIORI algorithm, although to implement it for very large databases we would need to be quite cautious to make it properly efficient.

Most of the algorithms that are in the literature are based on the Apriori algorithm. It only finds Boolean association rules not the quantitative ones. Its pseudo code is given in Figure 3. This algorithm makes a number of passes over the database. During pass k, the algorithm finds the set of frequent itemsets $L_k$ of length k that satisfy the minimum support requirement. The algorithm terminates when $L_k$ is empty. A pruning step eliminates any candidate, which has a smaller subset. Apriori uses specialized data structures to speed up counting and pruning ( Hash trees and Hash Tables, respectively). The user should refer to any book of data structures like [12] for further details.

1. **$L_1$ = { frequent 1-itemsets }**
2. **for ( k = 2; $L_k$-1 != Ø; k ++ )**
3. **$C_k$ = apriori_gen($L_k$-1);**
4. **for all transactions t element of D**
5. **subset($C_k$, t);**
6. **$L_k$ = { c element of $C_k$ | c.count > MinSup };**
7. **Answer = $U_k$ $L_k$;**

Fig 2.1 The Apriori Algorithm [3]

Initially $L_1$ contains all the items that satisfy minimum support. Then for k = 2, 3, 4,.. the algorithm generates $C_k$ of candidate items of length k using $L_{k-1}$. This is done in function *apriori_gen()*. Once the candidate itemsets are found, their frequencies are computed in function *subset()* by counting how many transactions contain these candidate itemsets. Time required to find the rules from candidate sets is relatively small when compared to time required to find candidate sets. **apriori_gen Function** [3]{*Candidate Generation*} This function takes as argument $L_{k-1}$ the set of all large (k-1) itemsets. It returns a superset of the set of all large k-itemsets. The function works as follows.

Next in the prune step, we delete all itemsets $c \in C_k$ such that some (k-1)-subsets of c is not in $L_{k-1}$:

Forall itemsets $c \in C_k$ do
  Forall (k-1)-subsets s of c do
    If ($s \notin L_{k-1}$) then
      Delete c from $C_k$;

**Subset Function** {*Counting Support of Candidates*}
This is the time consuming step in the algorithm Candidate itemsets are stored in a hash tree. A node of the hash tree either contains a list of itemsets (a leaf node) or a hash table (an interior node). In an interior node each bucket of hash table points to another node. The root of the hash tree is defined to be at depth 1. An interior node at depth d points to nodes at depth d+1. Itemsets are stored in the leaves. When we add an itemset c, we start from the root and go down the tree until we reach a leaf. At an interior node at depth d, we decide which branch to follow by applying a hash function

to the d Th item of the itemset. All nodes are initially created as leaf nodes. When the number of itemsets in a leaf node exceeds a specified threshold, the leaf node is converted in to an interior node.

Starting from the leaf node the subset function finds all the candidate contained in a transaction t as follows. If we are at a leaf, we find which of the itemset in the leaf are contained in t and add references to them to answer set. If we are at an interior node and we have reached it by hashing the item I, we hash on each item that comes after I in t and recursively apply this procedure to the node in corresponding bucket. For the root node, we hash on every item in t.

To see why the subset function returns the desired set of references, consider what happens at the root node. For any itemset c contained in transaction t, the first item of c must be in t. At the root, by hashing on every item in t, we ensure that we only ignore itemsets that start with an item not in t. Similar arguments apply at lower depths. The only additional factor is that since the items in any itemset are ordered, if we reach the current node by hashing the item I, we only need to consider the item in t that occur after i.

**Extended Apriori Algorithm** Starting with the frequent items, generates all frequent itemsets using an algorithm based on the Apriori algorithm. The proposed algorithm extends the candidate generation procedure to add pruning using the "greater-than-expected-value" interest measure, and uses a different data structure for counting candidates.

A third phase, namely Interest Prune Phase, is included in the apriori_gen function to prune uninteresting candidates. In addition to a hash tree , this algorithm uses R*-trees in counting support of candidates to speed up counting.
The main extension of this algorithm is that it finds associative rules containing both quantitative and Boolean attributes. And when it is necessary quantitative attributes can be partitioned into intervals.

**Counting Candidates of Multiple Sizes in One Pass [3]:**

Rather than counting only candidates of size k in $k_{th}$ pass, we can also count the candidates $C_{k+1}'$ , where $C_{k+1}'$ is generated from $C_k$, etc. Note that $C_{k+1}' \supseteq C_{k+1}$ since $C_{k+1}$ is generated from $L_k$. This variation can pay off in the later passes when the cost of counting and keeping in memory additional $C_{k+1}' - C_{k+1}$ candidates becomes less than the cost of scanning the database.

Thus the Apriori algorithm performs as many passes over the data as the maximum number of elements in a candidate itemset, checking at pass k the support for each of the candidate in $C_k$. The two important factors which govern the performance are number of passes made over all the data and efficiency of those passes.

To address both of these issues, an algorithm DIC (**D**ynamic **I**temset **C**ounting) was introduced which reduces the number of passes made over the data while keeping the number of itemsets which are counted in any pass relatively low as compared to methods based on sampling.

### 2.4.4 Dynamic Itemset Counting (DIC) Algorithm [5]:

The intuition behind DIC is that it works like a train running over the data which stops at intervals M transactions apart. (M is a parameter). When the train reaches the end of the transaction file, it has made one pass over the data and it starts over at the beginning for the next pass. The "passengers" on the train are itemsets. When an itemset is on the train, we count its occurrence in the transaction that are read.

If we consider Apriori in this metaphor, all itemsets must get on at the start of a pass and get off at the end. The 1-itemsets that take the first pass, the 2-itemsets take the second pass, and so on. See fig 2.2. In DIC, flexibility have been added to the itemsets to get on at any stop as long as they get off at the same stop the next time the train goes around. Therefore, the itemset has "seen" all the transactions in the file. This means that we can start counting an itemset as soon as we suspect it may be necessary to count it instead of waiting until the end of the previous pass.

For example if database contains 40,000 transactions and M=10,000, all the 1-itemsets will be counted in the first 40,000 transactions read. However, counting of 2-itemsets will begin after the first 10,000 transactions have been read. Similarly counting of 3-itemsets will begin after 20,000 transactions. For now we assume there are no 4-itemsets we need to count. Once end of file is reached, counting the 1-itemsets

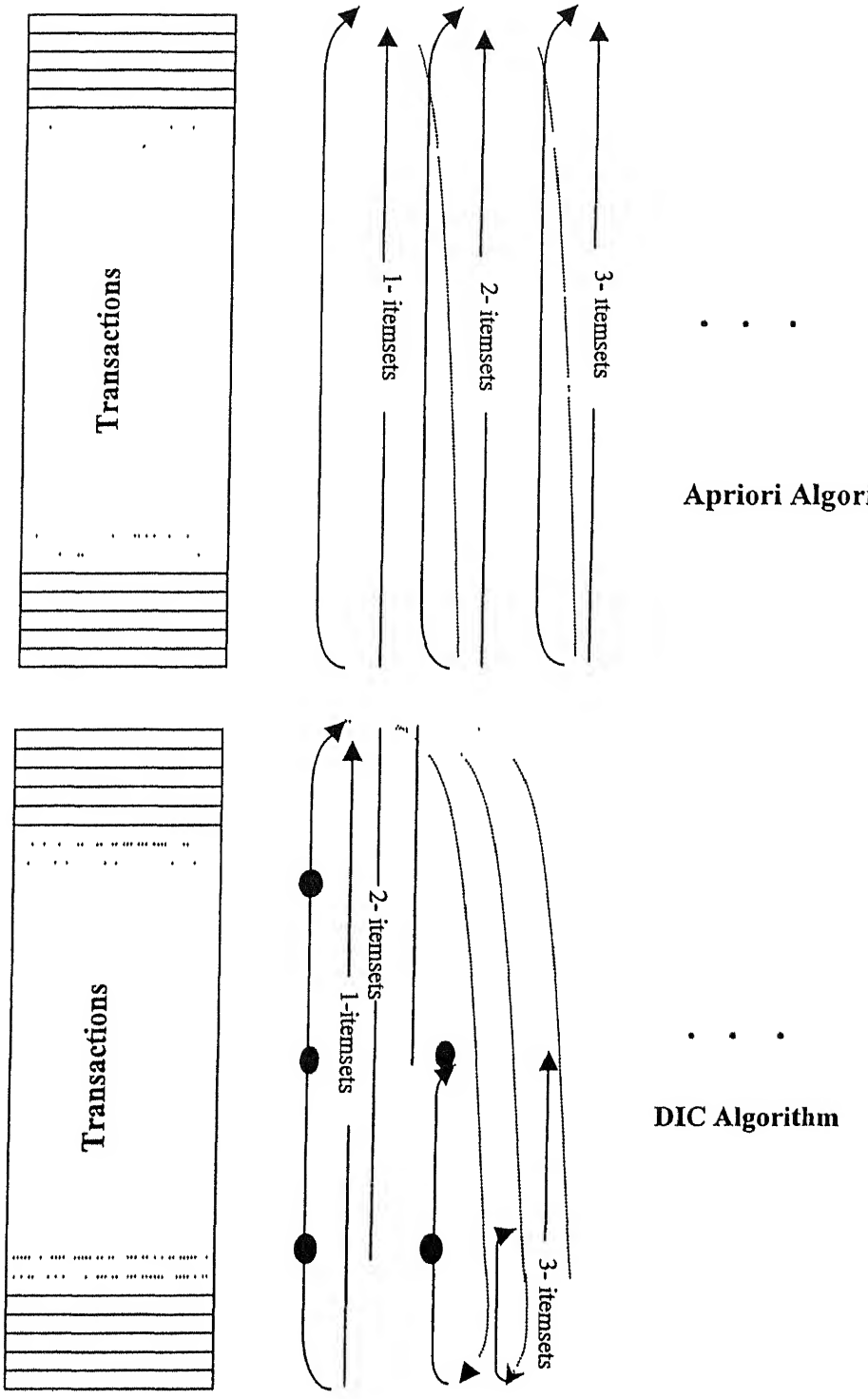


Fig 2.2
Counting itemsets in Apriori and DIC

is stopped and we go back to the start of the file to count 2 and 3-itemsets. After the first 10,000 transactions, we will finish counting the 2-itemsets and after 20,000 transactions, we will finish counting the 3-itemsets. In total, we have made 1.5 passes over the data instead of the 3 passes a level-wise algorithm would make

***Implication Rules:***

This measure which is based on conviction is supposed to be a more useful and intuitive measure than confidence and interest. Unlike confidence, conviction is normalized based on both the antecedents and the consequent of the rule like the statistical notion of correlation. Furthermore, unlike interest, it is directional and measures actual implication as opposed to the co-occurrence. Because of these two features, implication rules can produce useful and intuitive results on a wide variety of the data. For example, the rule past active duty in military⇒ no service in Vietnam has a very high confidence of 0.9. Yet it is clearly misleading since having past military service only increases the chances of having served in Vietnam. In tests on census data, the advantages of conviction over rules based on confidence and interest are evident.

**Counting Large Itemsets:**

Itemsets form a large lattice with the empty itemset at the bottom and the set of all items at the top (see fig2.3). Some itemsets are large (denoted by boxes), and the rest are small. Thus, in the example, the empty itemset, A, B, C, D, AB, AC, BC, BD, CD, ABC are large.

To show that the itemsets are large we can count them. In fact, we must, since we generally want to know the counts. However, it is infeasible to count all of the small itemsets. Fortunately, it is sufficient to count just the minimal ones (the itemsets that do not include any other small itemsets) since if an itemset is small, all of its supersets are small too. The minimal small itemsets are denoted by circles; in our examples AD and BCD are minimal small. They form the top side of the boundary between the large and small itemsets (in lattice theory the minimal small itemsets are called the prime implicants).

An algorithm that counts all the large itemsets must find and count all of the large itemsets and minimal small itemsets (i.e. all of the boxes and circles). The DIC algorithm, described here, marks the itemsets in four different possible ways:

- Solid Box --- confirmed large itemset — an itemset we have finished counting that exceeds the support threshold.
- Solid Circle--- confirmed small itemset --- an itemset we have finished counting that is below the support threshold.
- Dashed Box --- suspected large itemset --- an itemset we are still counting that exceeds the support threshold.
- Dashed circle --- suspected small itemset --- an itemset we are still counting that is below the support threshold.

The DIC algorithm works as follows:

1. The empty itemset is marked with a solid box. All the 1- itemsets are marked with dashed circles. All other itemsets are unmarked. See fig 2.4(a)
2. Read M transactions M can be any number like anything from 100 to 10,000. For each transaction, increment the respective counters for the itemsets marked with dashes.
3. If a dashed circle has a count that exceeds the support threshold, turn it into a dashed square. If any immediate superset of it has all of its subsets as solid or dashed squares, add a new counter for it and make it a dashed circle. See fig.2.4 (b) and (c).
4. If a dashed itemset has been counted through all the transactions, make it solid and stop counting it.
5. If we are at the end of the transaction file, rewind to the beginning. See fig 2.4 (d)
6. If any dashed itemsets remain, go to step 2.

***Significance of DIC:***

There are a number of benefits to DIC. The main one is the performance. If the data is fairly homogeneous throughout the file and the interval M is reasonably small, this algorithm generally makes on the order of two passes. This makes the algorithm considerably faster than Apriori which must make as many passes as the maximum size of a candidate itemset. If the data is not fairly homogeneous, we can run through it in a random order.

Besides performance, DIC provides considerable flexibility by having the ability to add and delete counted itemsets on the fly. As a result, DIC can be extended to parallel and incremental update versions.

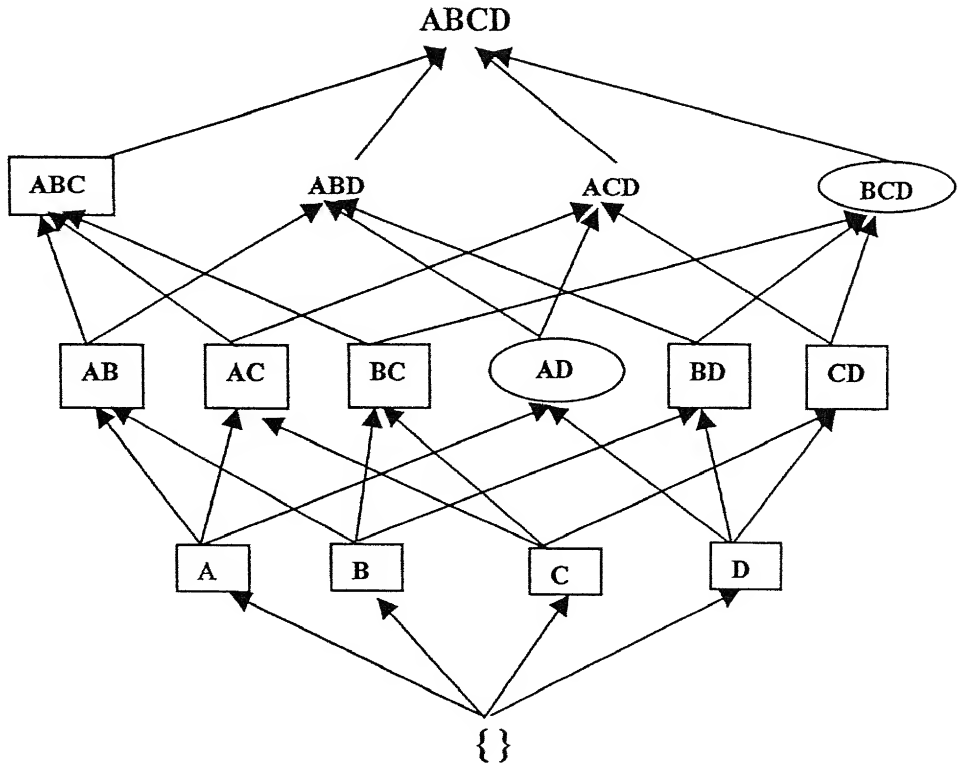


**Fig. 2.3 An itemset Lattice**

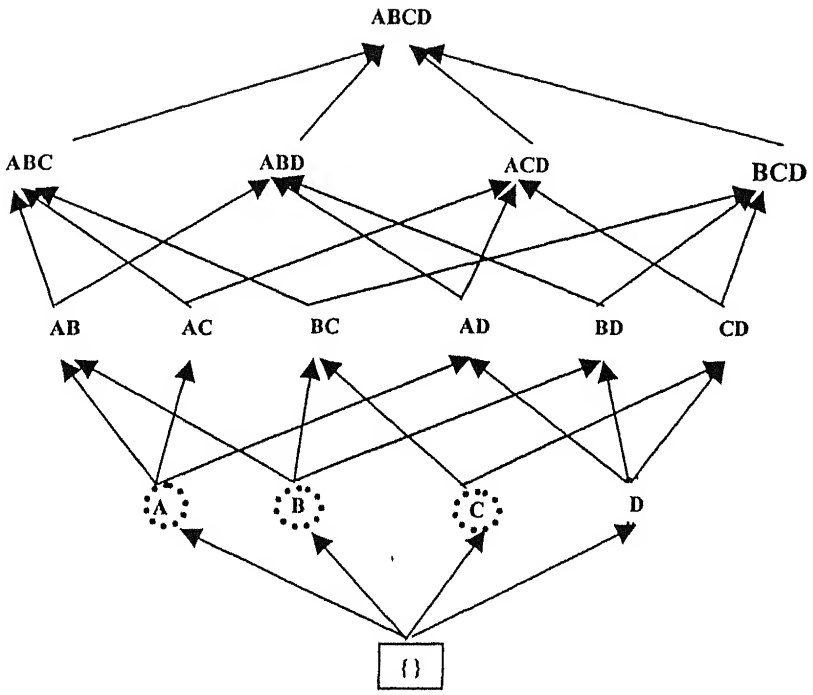


Fig 2.4(a) Start of DIC algorithm

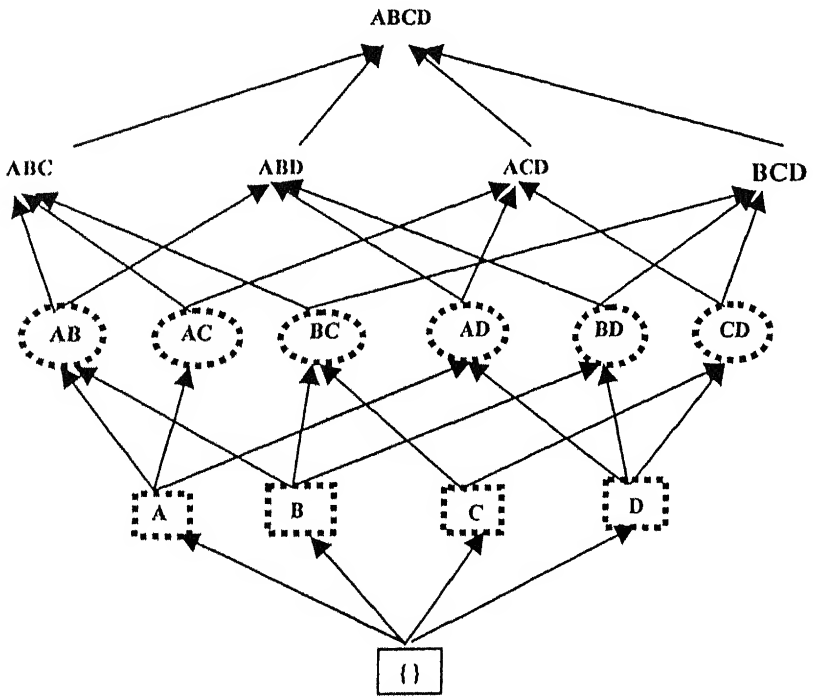


Fig 2.4(b) After M transactions

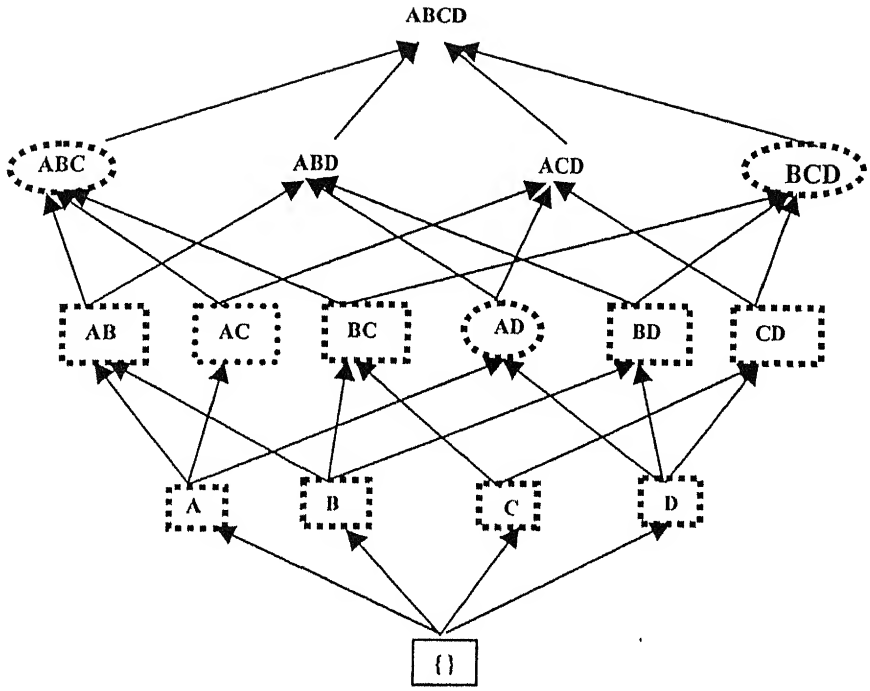


Fig 2.4( c) After 2M Transactions

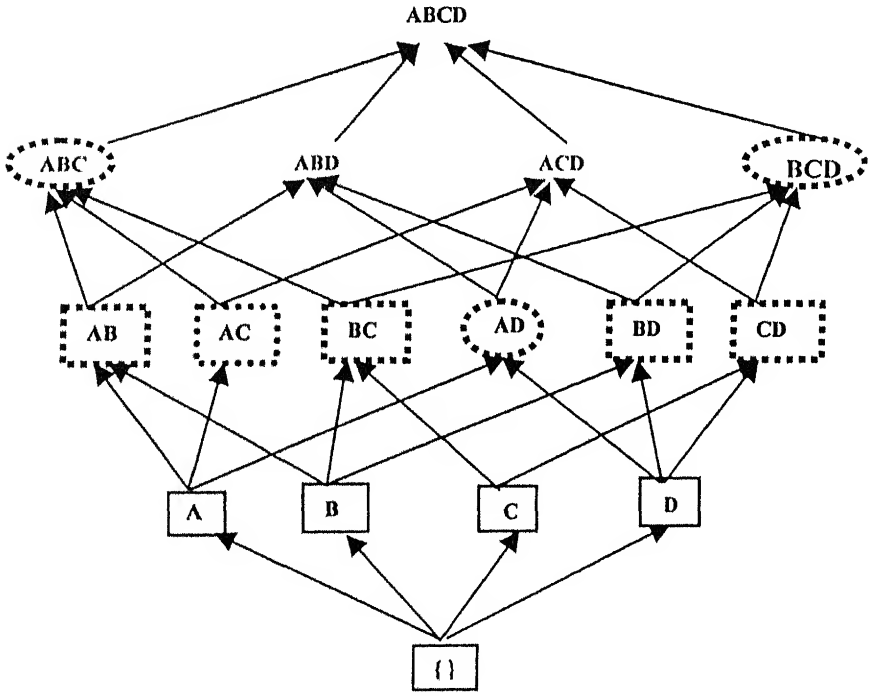


Fig 2.4(d) After one pass

### 2.4.5 Algorithms based on Vertical Data Layout ( TID- List based approach) [8]:

As already mentioned the vertical data layout consists of two parts. One contains the item and other contains the transaction ids in which that particular item has appeared. If the original database is horizontal one then one can scan the database once and can get the tid-lists for all 1-itemsets. the tid-lists for all other higher order itemsets can be found by intersecting the tid-lists at the previous level. After finishing the tid-list generation at any level we check whether or not that particular itemset is large. If the itemset is small then we need not to count any higher order itemset containing this itemset.

Lemma1: *All subsets of a frequent itemset are frequent*

A corollary of this lemma is that *all supersets of an infrequent itemset are infrequent* This observation forms the basis of a very strong pruning strategy in a bottom-up search procedure for frequent itemsets, which has been leveraged in many association mining algorithms. Namely only the itemsets found to be frequent at the previous level need to be extended as candidates for the current level.

Lemma2: *The maximal frequent itemsets uniquely determine all frequent itemsets*

This observation tells us that our goal should be to devise a search procedure that quickly identifies the maximal frequent itemsets.

Lemma3: *Let X and Y be two itemsets, with* $X \subseteq Y$. *Then* $\Gamma(X) \supseteq \Gamma(Y)$

This lemma states that if X is a subset of Y, then the cardinality of the tid-list of Y (i.e., its support) must be less than or equal to the cardinality of the tid-list of X. A practical and important consequence of the above lemma is that the cardinalities of the intermediate tid-lists shrink as we move up the lattice. This results in very fast intersection and support counting.

### 2.4.6 Other algorithms:

There are some other algorithms proposed in literature for the association rule mining. These are Dynamic Hashing and Pruning(DHP), Partition, SEAR and Spear and finally Eclat, MaxEclat, Clique, and MaxClique. We will have a quick tour to these algorithms:

***Dynamic Hashing and Pruning:*** [4,13]The DHP algorithm proposed by Jong Soo Park and colleagues extends the Apriori approach by using a hash table to precompute approximate support of 2-itemsets during the first iteration. The second iteration need count only those candidates falling in hash cells with minimum support. This hash table technique can successfully remove many candidate pairs that would eventually have become infrequent.

***Partition:*** [13]Ashok Savasere and others proposed two-pass partition algorithm, which logically divides the horizontal database into non-overlapping partitions. Each partition is read, and vertical tidlists (lists of all tids where the item appears) are formed for each item. Partition then generates all locally frequent itemsets through tid-lists intersections. Locally frequent itemsets from all partitions merge to form a global candidate set. Partition then makes a second pass over all the partitions and obtains all candidates' global counts through tid-list intersections.

***Eclat, MaxEclat, Clique and MaxClique:*** [8,13] A completely different design characterizes the equivalece class-based algorithms proposed by Rakesh Agrawal and colleagues. The simplest is Eclat; the best is MaxClique. These methods use a vertical database format, complete search, and a mix of bottom-up and hybrid search, and they generate a mix of maximal and nonmaximal frequent itemsets.

The main advantage of using a vertical format is that we can determine the support of any k-itemset by simply intersecting the tid-lists of the lexicographically first two (k-1)-length subsets that share a common prefix( the generating itemsets). These methods break the large search space into small, independent, manageable chunks. These chunks can be processed in memory through prefix or clique based equivalence classes; the clique based approach produces much smaller classes. Each class is independent in that it has complete information for generating all frequent itemsets that share the same prefix.

### 2.4.7 Parallel and Distributed Mining of Association Rules [13]:

Researchers expect parallelism to relieve current ARM methods from the sequential bottleneck, providing scalability to massive data sets and improving response time. Achieving good performance on today's multiprocessor systems is not trivial. The main challenges include synchronization and communication minimization, workload balancing, finding good data layout and data decomposition and disk I/O minimization (which is specially important for ARM). The parallel design space spans three main components; the hardware platform, the type of parallelism, and the load balancing strategy.

- ***Distributed vs. Shared Memory systems:***

Two dominant approaches for using multiple processors have emerged: distributed memory (Where each processor has a private memory) and shared memory (where all processors access common memory). A shared-memory (SMP) architecture has many desirable properties. Each processor has direct and equal access to all system's memory. Parallel programs are easy to implement on such systems.

A different approach to multiprocessing is to build a system from many units, each containing a processor and memory. In distributed memory environment, each processor has its own local memory, which only that processor can access directly. For a processor to access the data in the local memory of another processor, message passing must send a copy of the desired data elements from one processor to another. Although a shared memory architecture offers programming simplicity, a common bus's finite bandwidth can limit scalability. A distributed memory, message passing architecture cures the scalability problem by eliminating the bus, but at the expense of programming simplicity.

A third, very popular paradigm combines the best of the distributed and shared memory approaches. Included in this paradigm are hardware –or software- distributed shared memory systems. These systems distributed shared –memory systems. These systems distribute the physical memory among the nodes but provide a shared global address space on each processor. The hardware or software ensures cache coherence; so locally cached data always reflects any processor's latest modification. Clusters of SMP workstations (clumps) are also part of this mixed paradigm. Clumps necessitate a hierarchical parallelism approach, with SMP primitives used in a node and message passing used among the SMP nodes.

The performance optimization objectives for distributed-memory machines versus shared-memory systems depend on the underlying architecture. In DMMs synchronization is implicit in message passing, so the goal becomes communication optimization. For SMPs synchronization stems from locks and barriers, and the goal is to minimize these points. Data decomposition is very important for distributed memory, but not for shared memory. While parallel I/O comes free in DMMs, it can be problematic for SMP machines, which typically serialize I/O. the main challenge for obtaining good performance on DMMs is finding a good data decomposition among the nodes and minimizing communication.

For SMPs,the objective is to achieve good data locality. This means we must maximize the ping-pong effect, where multiple processor might be trying to modify different variables that coincidentally reside on the same chache line.

- **Data vs. Task Parallelism:**

Task and data parallelism are the two main paradigms for exploiting algorithm parallelism. For ARM, data parallelism corresponds to the case where the database is partitioned among p processors—logically partitioned for SMP s, physically partitioned for DMMs. Each processor works on its local partition of the database but performs the same computation of counting support for the global candidate itemsets. Task parallelism corresponds to the case where the processors perform different computations independently, such as counting a disjoint set of candidate, but have or need access to the entire data, but for DMMs, the process of accessing the database can involve selective replication or explicit communication of the local portions. Hybrid parallelism, which combines bith task and data parallelism, is also possible and perhaps desirable for exploiting all available parallelism in ARM methods.

- **Static Vs Dynamic Load Balancing:**

Static load balancing initially partitions work among the processors using a heuristic function; no subsequent data or computation movement is available to correct load imbalances resulting from ARM algorithms' dynamic nature. Dynamic load balancing seeks to address this by taking work from heavily loaded processors and reassigning it to lightly loaded ones. Computation movement also entails the data movement, because the processor responsible for a computational task needs the data associated with that task. Dynamic load balancing thus incurs additional costs for work and data movement, and also for the mechanism used to detect whether there is an

imbalance. However, dynamic load balancing is essential if there is a large load imbalance or if the load changes with time.

Dynamic load balancing is especially important in multi-user environments with transient loads and in heterogeneous platforms, which have different processor and network speeds. These kinds of environments include parallel servers and heterogeneous clusters, metaclusters, and superclusters ( the so called grid platforms that are becoming common today). All extant ARM algorithms use only static load balancing that is inherent in the initial partitioning of the database among the available nodes. This is because they assume a dedicated, homogeneous environment.

The main design issues in DMMs are minimizing communication and evenly distributed data for good load balancing. The distributed-memory ARM algorithms that I now consider assume the database is partitioned among all the processors in equal-sized locks residing on each processor's local disk.

Two parallel formulations of apriori algorithm were proposed by R. Agarwal and John C. Shafer, *count distribution* (CD) and *data distribution* (DD). The CD algorithm scales linearly and has excellent speedup and sizeup behavior with respect to the number of treansactions. However, there are two problems with this algorithm. First, it does not parallelize the computation for building the hash tree. On a serial algorithm, this step takes relatively small amount of time. But on parallel computations, it can become a major bottleneck. Second, if the hash tree does not fit in the main memory, then the extra disk I/O for the multiple passes over the transaction database can be expensive on machines with slow I/O systems. Hence, the CD algorithm, like its sequential counterpart Apriori, is unscalable with respect to the increasing size of the candidate set. The DD algorithm addresses these problems of the CD algorithm by partitioning the candidate set and assigning a partition to each processor in the system. However, this algorithm suffers from three types of inefficiency. First, the algorithm results in high communication overhead due to an inefficient scheme used for data movement. Second, the schedule for interactions among processors is such that it can cause processors to idle. Third, each transaction has to be processed against multiple hash trees causing redundant computations.

Addressing these problems Eui-Hong Han, George Karypis and Vipin Kumar presented two new parallel formulations of Apriori Algorithm for mining association rules. These are intelligent Data Distribution (IDD) and Hybrid Distribution (HD). IDD

improves upon the DD algorithm by minimizing communication overhead and processor idling time, and by eliminating redundant computation. However, the static partitioning of the hash tree results in load imbalance that becomes severe for a large number of processors. Furthermore, even with the optimized communication scheme, the communication overhead of IDD grows linearly with the number of transactions. HD combines the advantages of both the CD algorithm and the IDD algorithm by dynamically grouping processors and partitioning the candidate set accordingly to maintain good load balance.

# Chapter-3

## Implementation and analysis of the ARM Algorithms

We implemented three existing algorithms for association rule mining described in chapter 2 subsections 2.4.3, 2.4.4, 2.4.5 using C/C++ and java (DIC and Apriori) and studied their performance for the synthetic data we generated. The implementation of any algorithm requires a suitable data structure. Here while discussing the implementation details we will pay extra attention to this aspect.

### *3.1 DIC Algorithm:*

The implementation of the DIC algorithm requires a data structure that can keep track of itemsets of interest. In particular, it must facilitate the following operations:

1. Add new itemsets to the list of the frequent itemsets.
2. Maintain a counter for every itemset. When any transactions is read, increment the counters of all active itemsets [see sec 2.4.4], which occur in the transaction. This must be very fast as it is the bottleneck of the whole process.
3. Maintain itemset status by managing transitions from active to counted (dashed to small) and from small to large (circle to square). Detect when these transitions should occur.
4. When itemsets do become large, determine what new itemsets should be added as dashed circles since they could now potentially be large.

We used a trie structure for this algorithm. The trie has following properties. Itemsets are sorted by their items. Every itemset has a node associated with it, as do all of its prefixes. The empty itemset is the root node. All the 1-itemsets are attached to the root node, and their branches are labeled by the item they represent. A trie having four items A, B, C, D is shown on the next page in fig 3.1.

The trie structure at its nodes store the following information. This structure is almost same for all algorithms with minor modifications. A node **Tr** consists of

1. **Item key** : char
2. **Item count** : integer initialized to zero

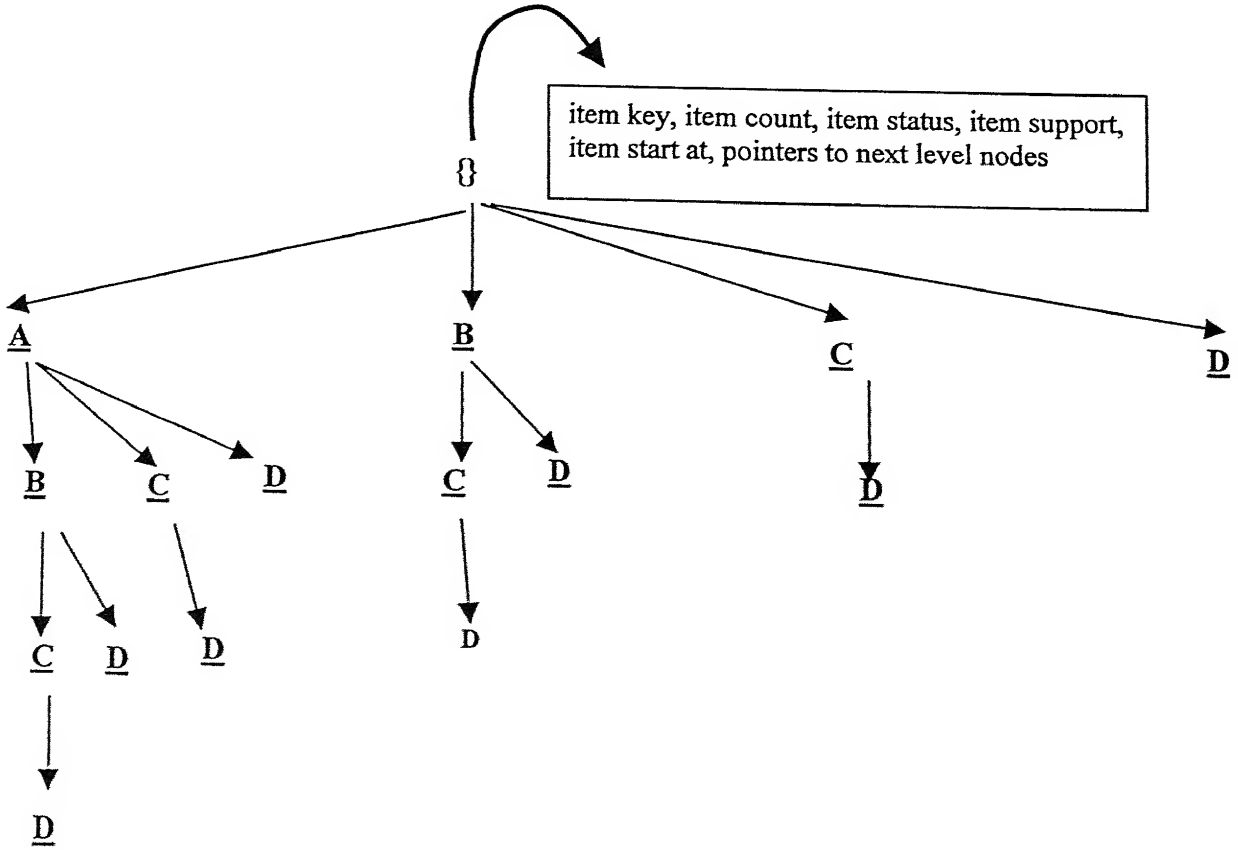


Fig 3.1

A Trie with four items A, B, C, D

3 **Item support** : float initialized to zero

4. **Item status**

*One of the following*

**DC (dashed Circle)** : integer flag value 1

**DS (dashed square)** : integer flag value 2

**SC (solid Circle)** : integer flag value 3

**SS (solid square)** : integer flag value 4

5. **Item count start pointer** : integer
6. **Pointers to next level nodes** ( referred to as **branch[ ]**)

### 3.1.1 Incrementing the appropriate Counters:

To increment the appropriate counters when reading an itemset is also a very interesting issue.

Suppose we have a transaction, T (with items T[1],T[2],.....,T[n]) in a certain order. To increment the appropriate counters we do the following, starting at the root node of the trie Tr:

```
Increment (Tr,T) {
        Tr.counter ++;
        If (Tr is not a leaf)
        for(i=0;i<n;i++)
         {
              if(Tr.branch[T[i]])
              increment(Tr.branch[T[i]], T[i+1.....n];
        }
        return;
    }
```

### 3.1.2 Analysis of the DIC algorithm:

One weakness of DIC is that it is sensitive to how homogeneous the data is. In particular, if the data is very correlated, we may not realize that an itemset is actually large until we have counted it in most of the database. If this happens, then we will not

shift our hypothetical boundary and start counting some of the itemsets' supersets until we have finished counting the itemset. To test this effect we randomized the data and re ran the DIC.

The algorithm checks the support of the itemsets after reading M transactions that is before adding itemsets of the next level. It adds only those itemsets at next level which have all their subsets as large itemsets. It means that depending upon M transactions only, we decide which itemsets to ignore and which one to take at the next level. If the data is not homogeneous then those M transactions may not give the true picture of the distribution of the itemsets in whole database. We found that this problem also depends on the value of M. When M approaches total number of transactions in the database DIC approaches to Apriori. In other words number of scans are equal to maximum number of items present in any transaction as in case of Aprioi. Where as when M is kept very low then there are considerable overheads and again the algorithm takes a lot of time. So one has to somehow find a proper value of M.

Fig. 3.2 shows the performance of this algorithm obtained by varying the value of M for original data and randomized data.

## 3.2 Apriori Algorithm:

There is not much difference in the program for Apriori and DIC. The only difference is that in place of the parameter M we scan the complete database for itemsets of any particular length. So M in this case is equal to total number of transactions.

### 3.2.1 Analysis:

The main problem with the Apriori algorithm is that it scans whole database as many times as the Maximum length of any itemset in the database. It takes a lot of time. The efficiency becomes very poor in case the database is too large to be accommodated in the main memory. We have considered this effect in chapter five.

## 3.3 TID-LIST based program for ARM:

As already been discussed, we used trie structure for all the algorithms. This trie structure proved to be very useful for the tid-list based algorithm. We stored the tid-list for an itemset at the node. Suppose the itemset is **ABC** then the tid list for this itemset will be stored at the node shown as filled square. Now suppose we want to have a tid

list for **ABCD** we can get it by intersecting the tid-list of **ABC** with that of **D** (see fig 3.3). Thus to get the tid list of any itemset we are required to intersect the tid-list of the previous node and the item that is recently added to get that particular itemset like ABC and D in the example.

Now we can see that this trie structure enables us to do all the calculations in only two intersections for each node. The pruning can also be very easily done using this structure. As soon as we get the tid-list for a particular itemset we determine the support for that particular itemset. If the support is greater than the support threshold then we continue growing downward otherwise we stop growing our trie beyond that itemset. For example suppose we have the support threshold as 0.5 or 50% and ABC has the support of only 0.4 then we will not allocate any memory for this node ABC and the trie will not grow beyond AB.

There can be different modules to implement different search procedures like bottom-up, top-down and hybrid. But the basic algorithm is same for all.

### 3.3.1 Analysis:

The algorithm is very simple and easy to implement but its efficiency depends upon the support threshold. When the support is very low then one is required to do $2^{(\text{no of items})}$ intersections. This becomes very much expensive. Fig 3.4 shows the performance of this algorithm. Even after pruning this is computationally inefficient as compared to other algorithms like Apriori and DIC. But at high value of support the algorithm gives comparable performance because of lesser number of intersection required.

One benefit of using this algorithm is that it can be used in the cases where the support is to be found for the combination of a few selected items only. For example if we have five items A, B, C, D, E and we are interested in computing the support for the combinations of A, B and D only then this algorithm performs superbly. Thus if we don't want to have a thorough analysis of the data and want to restrict ourselves to some selected items and combinations of them then we should use this algorithm. While taking a practical application of ARM on registration data of IIT Kanpur students in chapter 6, we will discuss the usefulness of this algorithm once more in this context.

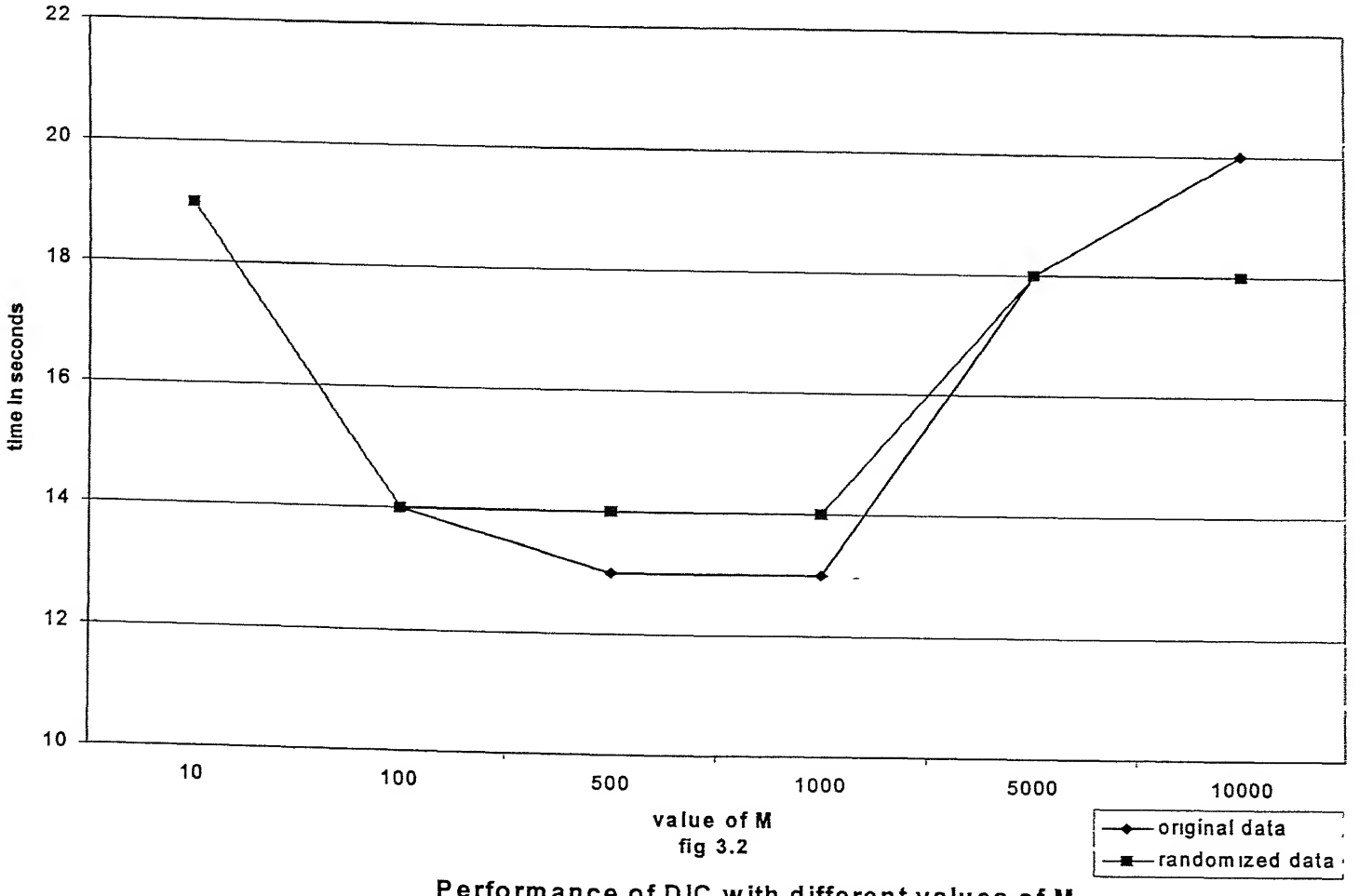


fig 3.2

Performance of DIC with different values of M
support threshold = 20% , Maximum number of items = 8, Number of transactions=10000

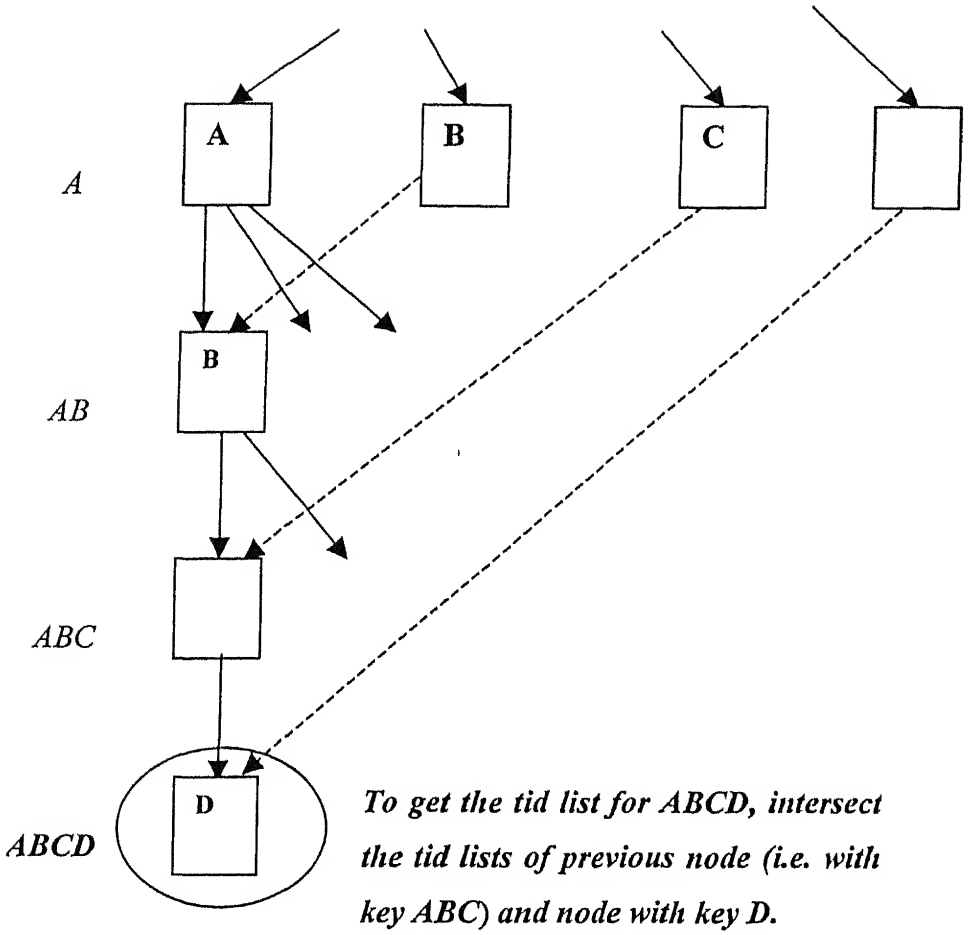


Fig 3.3

Strategy used for getting tid-lists of itemsets of next level

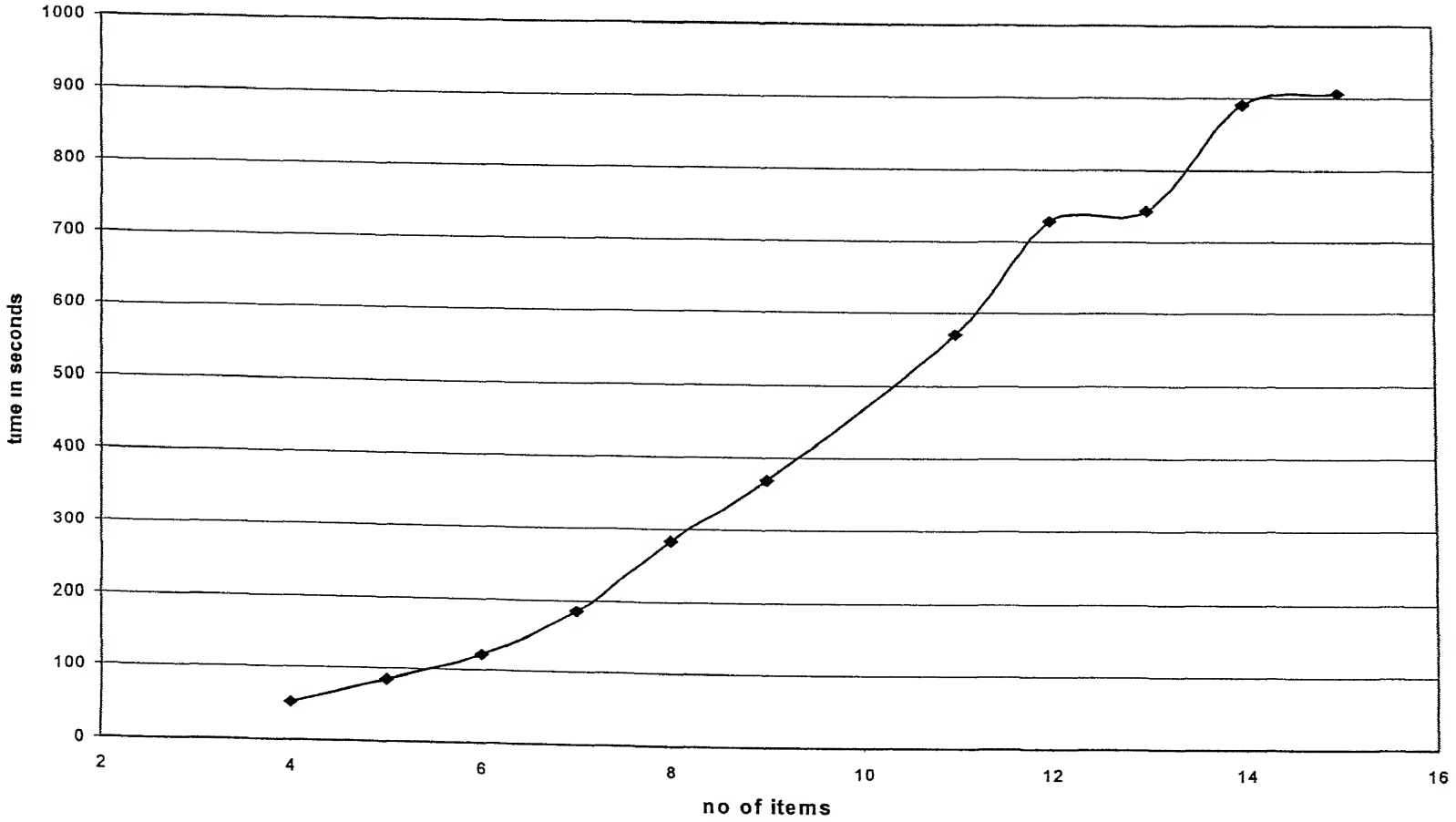


fig. 3.4
Performance of tid-list based program

### 3.4 Limitations in our implementation:

For simplicity we restricted ourselves to 26 branches at each node. The number 26 is taken to represent number of English alphabets. As we have seen that our synthetic data comprised of English alphabets we can directly relate our index of the branch to its key. The things become a bit simpler in this way. Though its not a big limitation and can be solved very easily but to understand the concepts of the ARM one can use this number.

### 3.5 Synthetic Data Generation:

To generate synthetic data we developed following algorithm:

The input to the program are.

1. Number of transactions required in the file (**total_tran**)
2. Maximum length of the itemsets (**Max_length**)
3. Possible Items ( like a,b,c...etc)

**Algorithm**

```
Count=0;
While( Count<total_tran)
    {
        Generate a random number from 1 to Max_length (x);
        for(i=0;i<x;i++)
        {
            Randomly select an alphabet (y);
            if(this alphabet is same as previously selected
        alphabet)
            i--;
            str[i]=y;
        }
        Sort the string thus generated of length x;
        Write it to the file;
        Count++;
    }
```

The data can be generated for any length of the itemsets. we generated the data with itemset length 4 to 15 and tested the algorithms on it.We have shown a sample page of the data thus generated in fig 3.5.

a
a c e g ı
a b c d e f g h ı j k
l
f ı
c d e f g k
a b c d e f g h j
a c d e f g ı j k l
b
b d f h ı k
a b c d e g h ı j k l

b c d e f g h i j l
e l
a c e f g h ı l
a f g h ı j k
c d e f g h ı k l
l
a d e f i j k l
a b c d e f g h i j k
l
b d e g ı
l
a
c f g
a b c e f l
e
i
a b c d e i k
c
a b c d e g h ı j k l
e f l
e h
b d h l
a b c e f g h ı k l
a c d g h j k l
a b c d e f g h ı j k
l
a b c d e f g h ı j k
l
a c e f h k l
c d e f g h ı j k
b c d f i k
a b c d e f h ı j k l
i l
c g h k
c f h j l
a b c d e f g h i j k
l
a c d e h l
a b c d e f g h i j k
l
a d f g h ı j k
e ı
b c d e f g ı j l
a c d
a b c e f l

Fig 3.5 synthetic data generated with 12 items

# Chapter-4

## Problems in handling large databases:

Mining association rules in large databases is still a challenging job. When the database is too large to be accommodated in main memory then the database is kept in the secondary memory. Mining task in these cases requires swapping the portions of the database into the core memory process it and then swapping it out of the core memory. So if we have very large database which can't be accommodated in core memory then we divide the database in the parts which are manageable and then processing them independently. After processing all of the portions we collate the results to find the global support. Fig. 4.1 shows the affect of this partitioning the database on the performance of the ARM program. We see that as the number of partition grows the time required for the ARM becomes large.

When we are doing ARM on line we are required to do it very fast. In most of such cases we are required to scan a huge database. Doing ARM on a single processor does not meet the real time constraint in these cases because the time taken by a single processor is very high and one cannot afford getting the information spending too much time In addition if the database is too large to be accommodated at one site then we distribute our data at different sites. In these cases distributed or parallel data mining is done.

One can follow any of the following methods depending upon the number of transactions and number of items for this purpose [8]:

1. Count Distribution
2. Data Distribution
3. Candidate Distribution

The **Count Distribution** algorithm is a simple parallelization of Aprioi (see fig 4.2). All the processors generate the entire candidate hash tree or trie from frequent k-1 itemset. Each processor thus independently gets partial supports of the candidates from its local database partition. Next, the algorithm does a sum reduction to obtain the global counts by exchanging local counts with all other processors. Rather than merging different hash trees, the algorithm needs to communicate only partial counts, because each processor has a copy of complete trie. Once the globally frequent

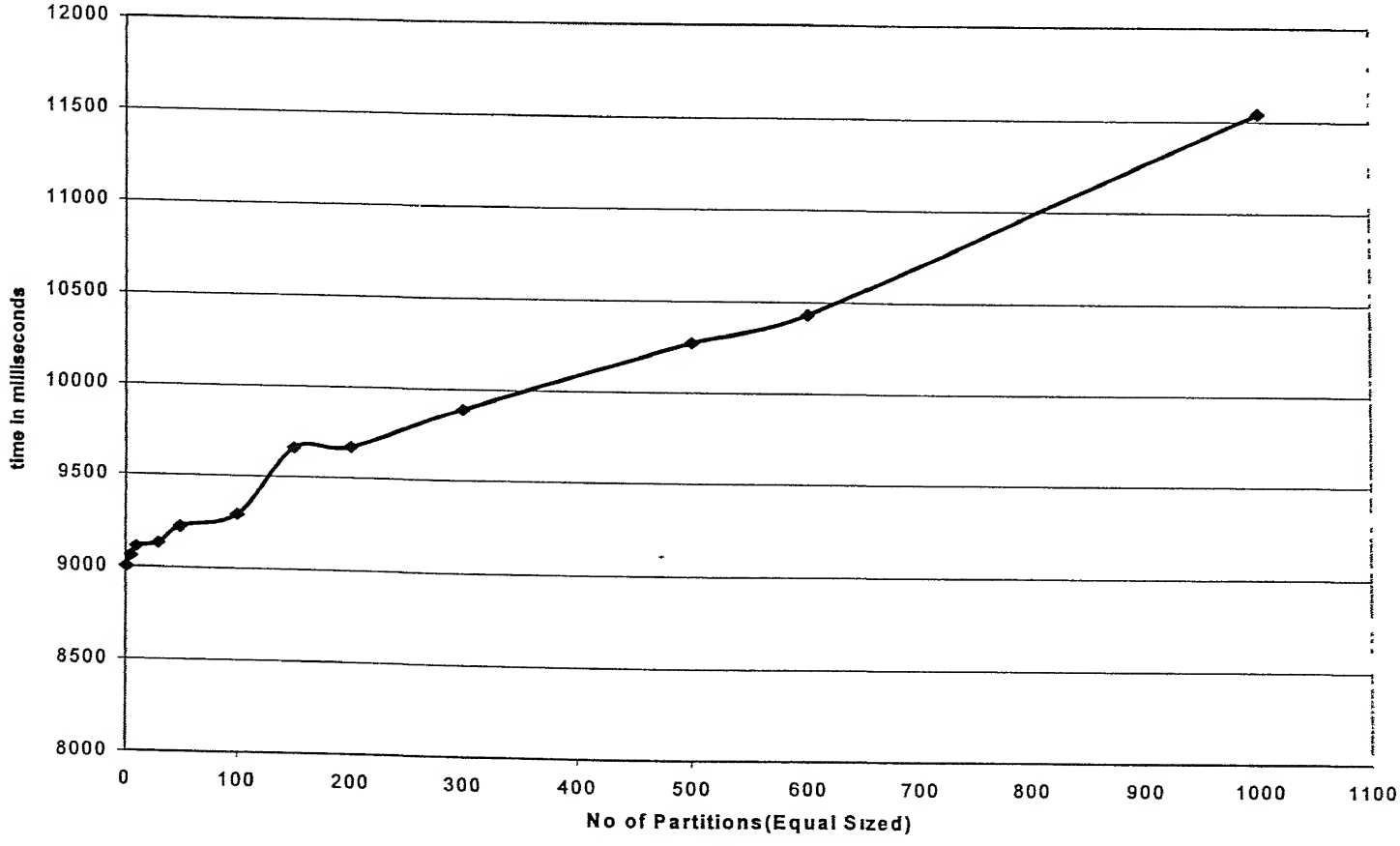


**fig. 4.1**
**Affect of partitioning the database acc. to the Core Memory size (effect on the efficiency of the program)**

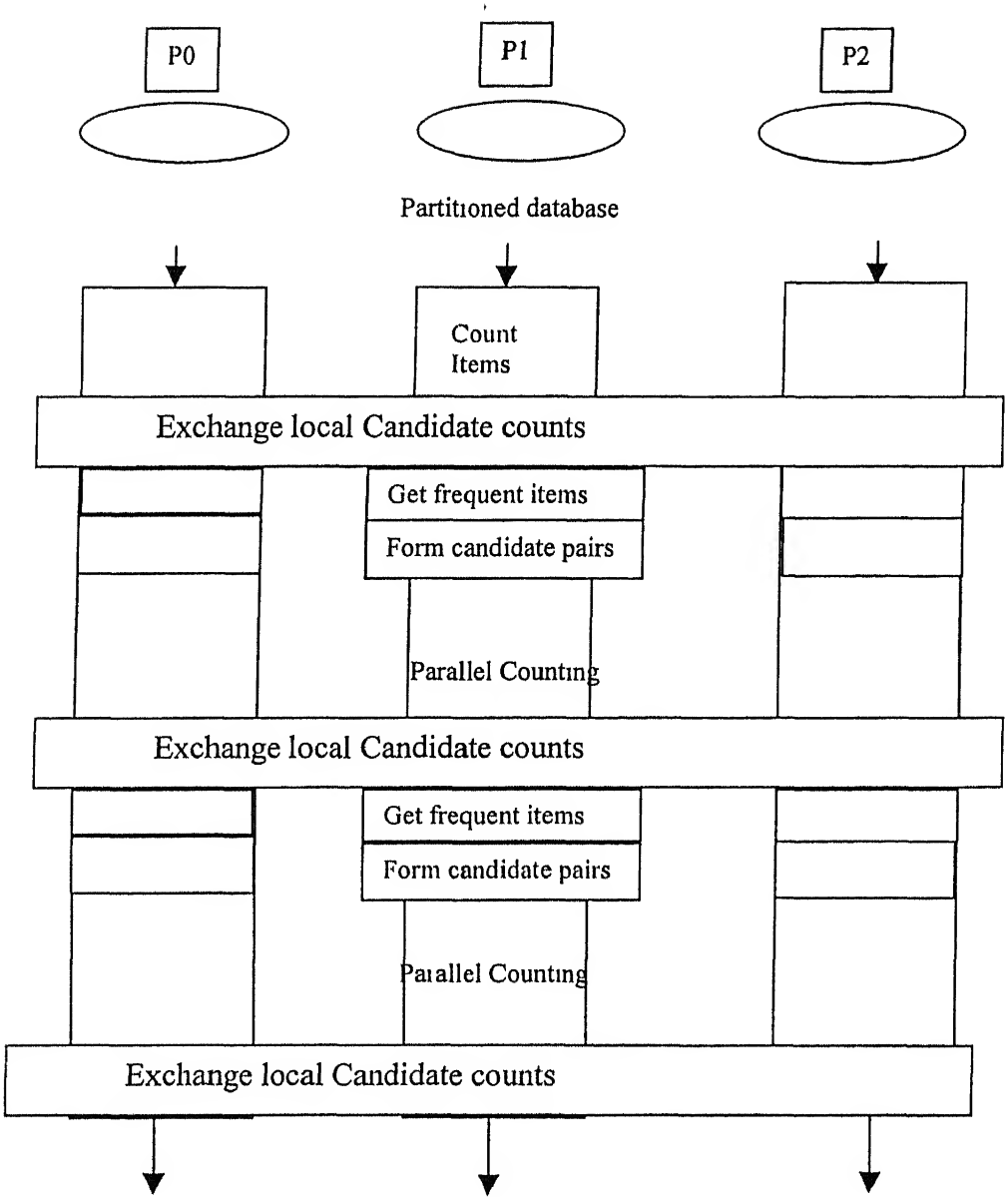


Fig. 4.2 Count Distribution

k-itemsets are obtained, each processor builds the entire candidate k+1-itemsets in parallel and repeats the process until all frequent itemsets are found [6,8].

This algorithm minimizes communication, because only the counts are exchanged among the processors. However, because the algorithm replicates the entire hash tree on each processor, it doesn't use the aggregate system memory effectively [13].

The **Data Distribution** algorithm [6] addresses the memory problem of the Count Distribution algorithm by partitioning the candidate item-sets among the processors. Each processor is responsible for computing the counts of its locally stored subset of the candidate item-sets for all the transactions in the database. Each processor scans the portions of the transactions assigned to the other processor as well as its locally stored portion of the transactions [8].

This algorithm exploits the total available memory better than CD as it partitions the candidate set among processor. As the number of processors increases, the number of candidate sets the algorithm can handle, also increases [8]. This algorithm suffers from high communication overhead and performs poorly when compared to count distribution [13] because during each pass of the algorithm, each processor sends to all other processors the portion of the database that resides locally.

The **Candidate Distribution** algorithm partitions the candidates during first iteration, so that each processor can generate disjoint candidates independent of other processors. The partitioning uses a heuristics based on support, so that each processor gets an equal amount of the work. At the same time, the database is selectively replicated so that a processor can generate global counts independently [6,8,13]. Candidate distribution performs worse than Count Distribution, because it pays the cost of redistributing the database while scanning the local database partitions repeatedly.

The count distribution algorithm exchanges the counts of all locally large itemsets at any iteration. When all the locally large itemsets are found then the global pruning is done to generate the candidate sets of the next level. Figure 4.2 shows an illustration of the count distribution.

*We now propose a method to increase the efficiency of the count distribution.*

Let a database have three portions distributed over three processors. Let it have 7 items {A, B, C, D, E , F, G}. If count distribution is applied to this example, it will require counting all the pairwise combinations of different items which will result in 21 candidates. The counts of all these 21 candidate sets are to be passed to find the global counts and to generate the candidate sets of next level i.e.candidate 3-itemsets. Now we will show how our approach reduces the candidate sets at each level.

**Local Pruning to Increase the Efficiency of Count Distribution:**

After first scan over the data in different portions we found that large 1-itemsets are {A, B, C, D, E, F, G}. of which A, B, C are large in first portion (say $P^1$), B,C,D are large in second portion (say $P^2$), and E, F, G are locally large in third portion say($P^3$). i.e.

*$HL_1^1$={A, B, C}

$HL_1^2$={B, C, D}

$HL_1^3$={E, F, G}

* *$Hl_k^i$ the set of large k-itemsets in the portion $P^i$*

Here is a simple observation:

**An itemset to be globally large should be large atleast at one of the sites and the set of global candidate itemsets for the next level can be generated from the union of the candidate itemsets at all the sites.**

It follows from above observation that the set of size-2 candidate sets in the portion 1 is equal to $CH_2^1$ ($CH_k^i$ denotes the set of candidate sets generated using $HL_k^i$) where $CH_2^1$=Apriori_gen($HL_1^1$)={AB, BC, AC} similarly, $CH_2^2$={BC, CD, BD}, and $CH_2^3$= {EF, FG, EG}. Hence, the set of candidate sets for large two-itemsets is $CH_2$= $CH_2^1$ U $CH_2^2$ U $CH_2^3$, and it has only eight candidates where as for the simple count distribution without any local pruning strategy we have to count 21 candidates and exchange their counts.

This shows that the technique is very effective in reducing the candidate sets.

So the over all procedure is as follows:

1. **Generate candidate k-itemsets at each site based on the large itemsets found at that site in k-1 iteration.**
2. **Scan the Local portion to find the local counts for each candidate set at each site.**
3. **Local Pruning:** ***For each candidate set at a site if it is not locally large then prune it. This candidate set which has been pruned at one site can be large at any other site and can also be globally large depending upon its counts at all the sites.***
4. **Each processor asks for the counts of the locally large candidate sets at its site from all other processors. Compute the global counts for them and find globally large itemsets.**
5. **Each processor sends a list of the globally large itemsets found at its site to all other processors. Thus each processor has a complete list of all the globally large itemsets for further processing.**

We could not implement this method for testing because of some hardware constraints. However intuitively one can see that the pruning will certainly increase the efficiency of the count distribution.

# Chapter-5

## Our New Approach for ARM (Game of Binary numbers):

We looked at the problem of association rule mining with a completely different view. Main problem in the ARM is that we usually handle a large database and that cannot fit completely in the main memory so we used to bring some portion of it into the main memory and then after doing the required operations on the data we swap it out of the memory. Thus the size of the portion of the database we can bring in is constrained by the available size of the main memory for this usage. One can easily see now that if we are able to compress the database some how, then we will be able to bring a bigger portion of the data in the main memory and can decrease the I/O time involved in this swapping in and swapping out of the portions considerably. First we will discuss the strategy we adopted for the compression of the data. This requires introductory knowledge of binary and decimal numbers

### 5.1 Method:

Let a transaction be represented by

| A | B | C | | E | F | | H |
|---|---|---|---|---|---|---|---|

----------5.1

equivalent to this, we can write

| 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

----------5.2

here one represents the presence of the item at that location and zero represents absence of that item. For example in the transaction given by 5.1, items D and G are absent, so we have written zero at the locations where they would be present if they were included in the transaction. One knows before hand what is the maximum number of distinct items. Like in our example it is eight. Thus we get a binary number corresponding to our original data representation of length eight.

**11101101**

If we store this in the file we will require eight bytes.

Now we convert it into its equivalent decimal number that comes out to be

$$1*2^0+0*2^1+1*2^2+1*2^3+0*2^4+1*2^5+1*2^6+1*2^7 = \mathbf{237}$$

if instead of writing transactions in the form shown in 5.1 or 5.2 in the data file we write 237 we will be able to save 5 bytes per transaction. The graph showing the

compression in the data is shown in fig 5.1. We have obtained the compression ratio in the range of 1.25 to 1.58 with almost monotonically increasing slope with number of items (see fig 5.2).

How this compression is going to help us? To answer this question let's suppose that size of main memory available to us is 80 kB. The original data file has a size of 100 kB and we are able to compress it using the method suggested by us up to 64 kB Now one can see that whole database can be brought into the main memory in this format and can be operated on in a single swapping in. Even if we are not able to do it in a single stroke we certainly get some improvement in the I/O time taken by the program for ARM.

Now we have a database in the form of a file which contains some numbers. We have to use those numbers to increment the counters of appropriate itemsets. The counters are stored in an array of structure **info_item** ( int count, float support) whose index represents the key of the itemset e.g. info_item[**1**] represents the structure for **a**. Alternatively we can say that index of the array count is the reverse decimal number for the binary number representation of the itemset. So there are $2^M$ elements in the structure info_item where M is the number of distinct items in the database. The information to be stored in the structure is the count and the support of the itemset. To increment appropriate counters we propose following method:

Let's take an example with M:

Suppose the original itemset is

| a | b | | d | | f | | |
|---|---|---|---|---|---|---|---|

this will be represented in binary representation as:

| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

so the binary number is 11010100

the equivalent ***reverse*** decimal number is ***43***.

The itemsets of which the counters are to be incremented are

**a, b, d, f, ab, ad, af, bd, bf, df, abd, abf, adf, bdf, abdf.**

So the file containing our database has entry 43 for this entry **abdf**. When we bring this file in the main memory we reconvert this entry into equivalent binary representation doing the reverse process i.e. 11010100.

Now we count the number of 1s present in the binary number thus obtained. Here it is **n=4.**

Store the weights of 1s in an array of integers of size n, W[n].

Thus

W[0]=1, W[1]=2, W[2]=8, W[3]=32;

We see that W is a vector of dimension n x 1.

$$W=\begin{bmatrix}1\\2\\8\\32\end{bmatrix}$$

Now generate the binary numbers from 1 to $2^n$ each of length n.

These are

0001,0010,0011,0100,0101,0110,0111,1000,**1001**,1010,1011,1100,1101,1110,1111.

Let these numbers be stored in an array of integers B[n].

For the bold faced number 1001 we have array elements as B[0]=1,B[1]=0,B[2]=0,B[3]=1.

Thus for all the binary numbers thus generated B is a vector of dimension 1x n.

B= **[1 0 0 1]**.

Multiply elements with same index stored in B[] and W[].

Thus for 1001 we get **W*B**

W*B = 1*1+0*2+0*8+1*32=33 => 10000100 => **a f**

so we get following numbers by multiplying B and W vectors.

| | B | W*B | | | |
|---|---|---|---|---|---|
| 1. | 0001 | 0*1+0*2+0*8+1*32=***32*** | => | 00000100 => | **f** |
| 2. | 0010 | 0*1+0*2+1*8+0*32=***8*** | | 00010000 | **d** |
| 3. | 0011 | 0*1+0*2+1*8+1*32=***40*** | | 00010100 | **d f** |
| 4. | 0100 | 0*1+1*2+0*8+0*32=***2*** | | 01000000 | **b** |
| 5. | 0101 | 0*1+1*2+0*8+1*32=***34*** | | 01000100 | **b f** |
| 6. | 0110 | 0*1+1*2+1*8+0*32=***10*** | | 01010000 | **b d** |
| 7. | 0111 | 0*1+1*2+1*8+1*32=***42*** | | 01010100 | **b d f** |
| **8.** | 1000 | 1*1+0*2+0*8+0*32=***1*** | | 10000000 | **a** |
| **9.** | 1001 | 1*1+0*2+0*8+1*32=***33*** | | 10000100 | **a f** |
| 10. | 1010 | 1*1+0*2+1*8+0*32=***9*** | | 10010000 | **a d** |

| | | | |
|---|---|---|---|
| 11 1011 | 1*1+0*2+1*8+1*32=*41* | 10010100 | **a d f** |
| 12. 1100 | 1*1+1*2+0*8+0*32=*3* | 11000000 | **a b** |
| 13. 1101 | 1*1+1*2+0*8+1*32=*35* | 11000100 | **a b f** |
| 14. 1110 | 1*1+1*2+1*8+0*32=*11* | 11010000 | **a b d** |
| 15. 1111 | 1*1+1*2+1*8+1*32=*43* | 11010100 | **a b d f** |

The indices at which the counters are to be incremented are as follows:

1, 2 , 3, 8, 9, 10, 11, 32, 33, 34, 35, 40, 41,42, 43

This algorithm proves to be very fast as compared to other algorithms.

Thus we can see that the data structure is also very much simplified using this method.

So steps in this algorithm are as follows:

1. **Store the data in secondary memory in decimal format**
2. **Bring this data into Main memory**
3. **Read one transaction and reconvert the number read into binary representation**
4. **Find the counters to be incremented using the method described above and increment it.**
5. **Read the next transaction.**
6. **Go to 5.**

## 5.2 Analysis:

fig. 5.1 shows the reduction in the size of database after the format conversion. From fig. 5.2 it is clear that the compression ratio is increasing with the number of items. Thus we can conclude that we can get better compression as the number of items will grow. This compression reduces the time spent in the swapping in and out of the portions of the database in case whole database can not fit in the main memory by a factor of compression ratio.

The simplification of the data structure enables us to do the mining faster than Apriori. The performance of the algorithm is shown in fig 5.3 and the comparison of the performance of our algorithm with Apriori is shown in fig 5.4.

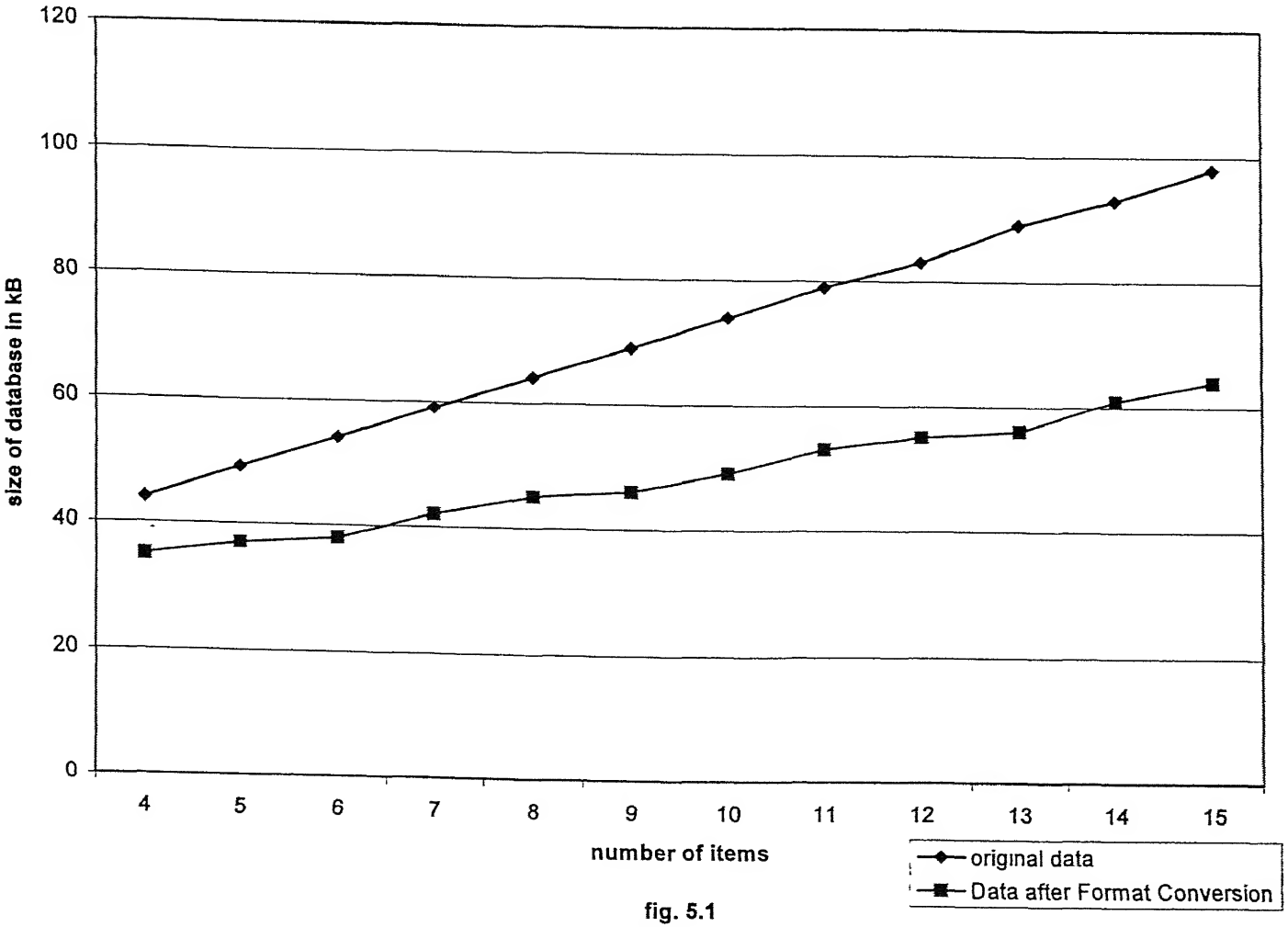


fig. 5.1

Comparative size of Data before and after format conversion

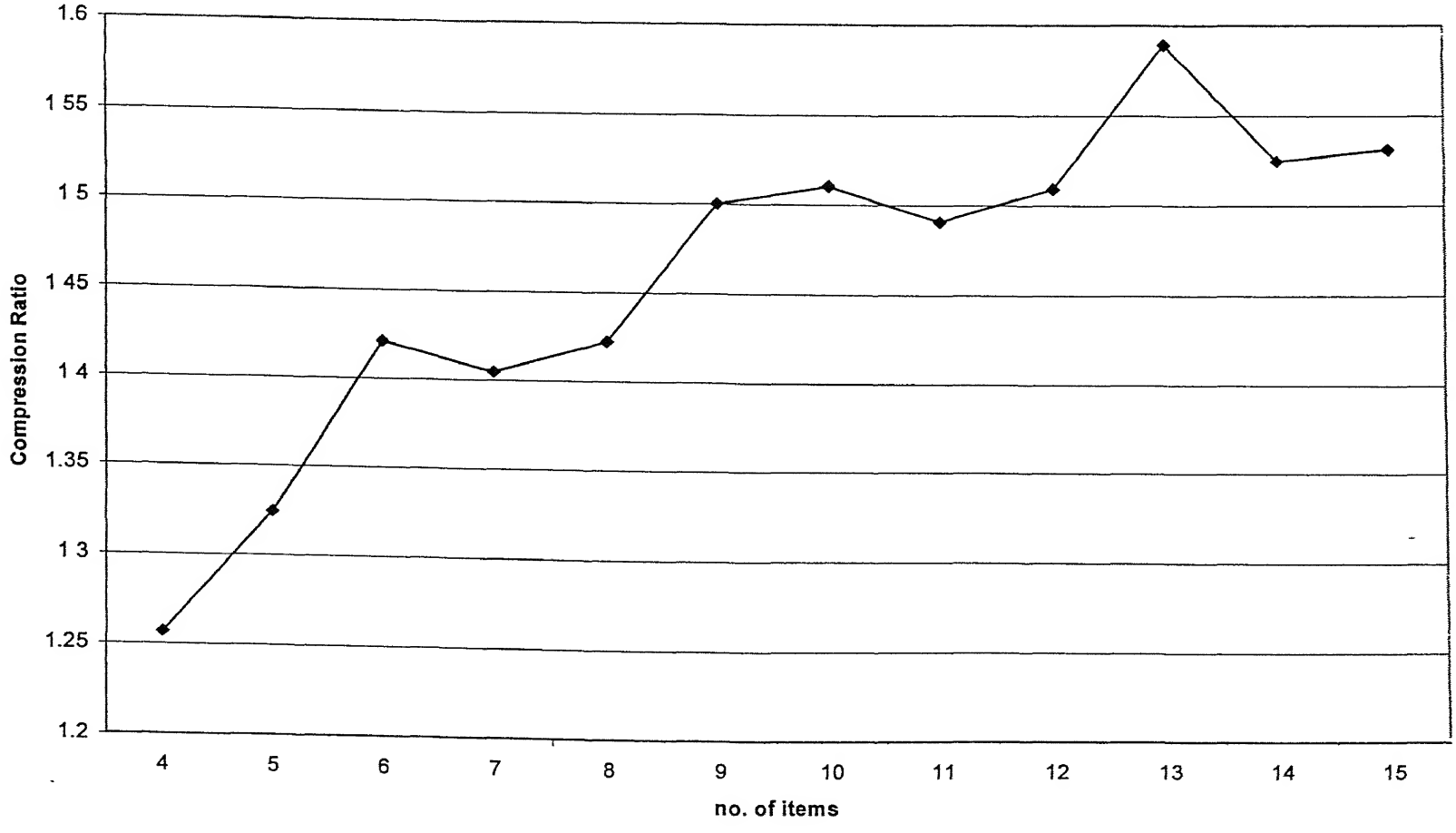


fig 5.2

Compression Ratio obtained after format conversion

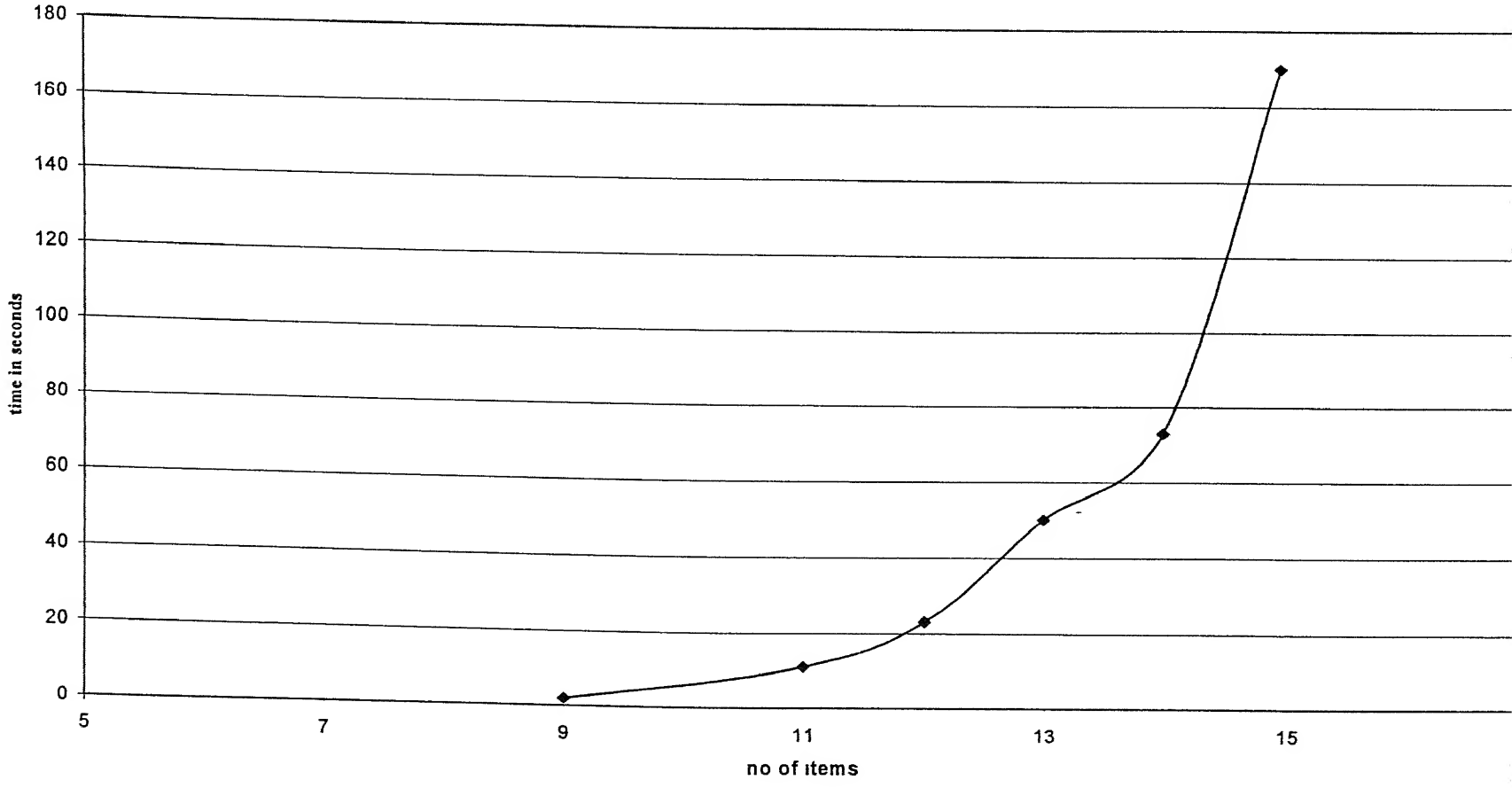


fig 5.3

performance of My_mine

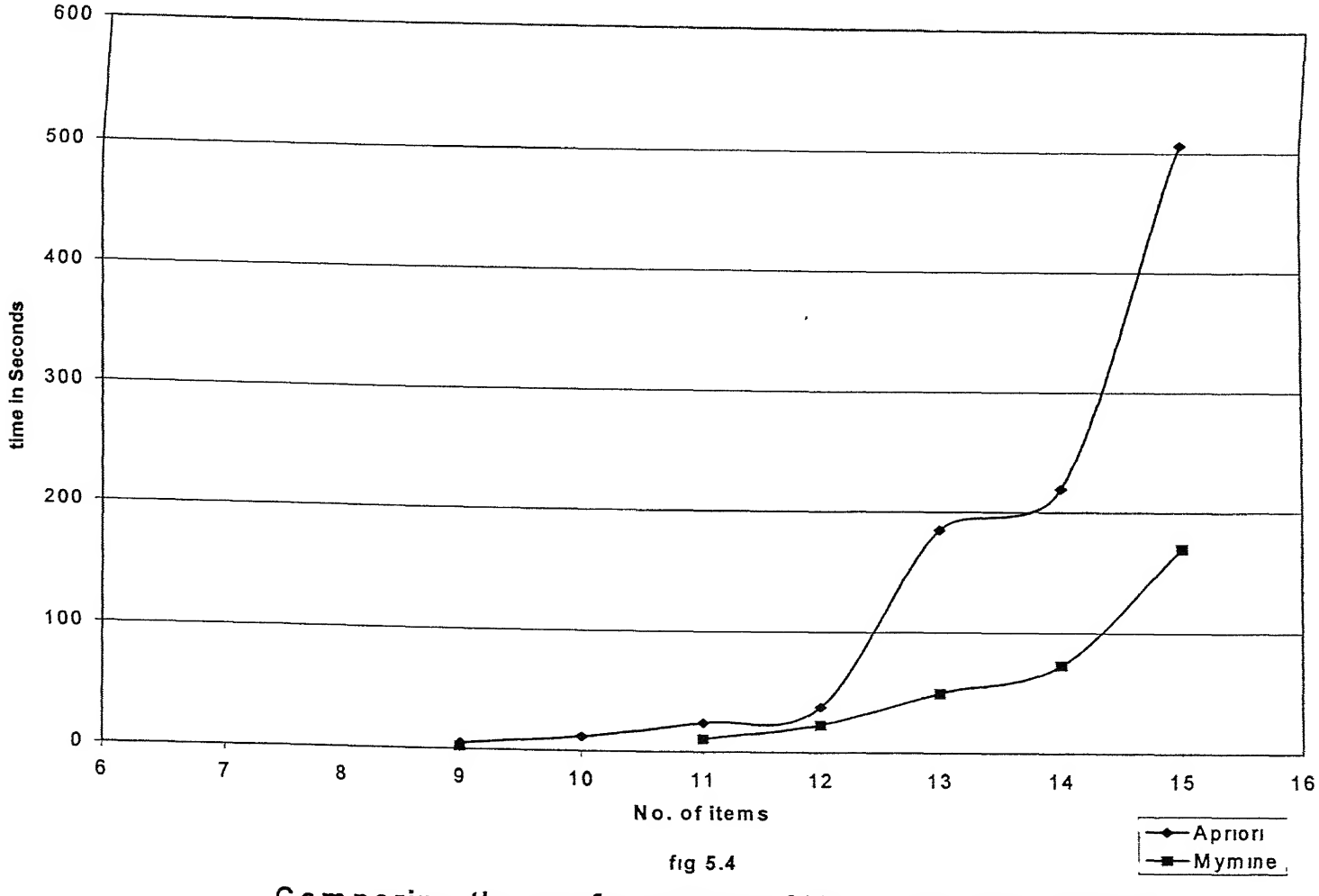


fig 5.4

**Comparing the performance of My_mine and Apriori**

# Chapter-6

# PRACTICAL APPLICATION OF ARM TECHNIQUES:

**6.1 Motivation:** We looked at the problem of scheduling electives offered in a particular semester which are open to the students of all the branches. Since students from different departments opt for these open electives it is very difficult to place these courses in a slot without any clash. In case of other courses we know before hand about the number of students opting for that course but in case of these open electives, since there is greater flexibility for the students in terms of Add/Drop we most of the time don't know how many students and from which departments the students are taking that particular course. So to decide the schedule for it is really a difficult task.

We used the concept of association rule mining for this task. ARM is not a tool to solve the problem completely but it helps us in finding out some interesting patterns which can be used to decide the schedule. Like if combination of two courses is having very large support then we should not place those two courses in same slot. We can also get some picture as how the students are inclined towards the courses of other departments.

## 6.2 Registration Data of IITK Students:

Students used to register for a semester before the start of next semester; i.e. in November for Spring semester and in April for Autumn semester. Students fill a registration form where they opt for the core courses and electives based on the requirements and their interests. They used to get about two months to decide the electives. The registration data includes:

1. Roll_No
2. Name
3. Category
4. Department
5. Courses Taken

We obtained the data from Office of Dean of Academic Affairs (DOAA). Some of the data was available as soft copy and other as hard copy. Finally we had two files of data containing 1200 data-sets (One file for data before Add/Drop and other for data after Add/Drop). The format for the file is as follows:

**S_no Roll_no Name course1 course2 course3.....**

**Dept. Category**

A sample page of the data is as follows:

```
    1 94133     MANOJ KUMAR VERMA          ME222   R   ME460   R   MTH203
    PSY151 S   PSY468

                ME    SC      SLOW-B

    2 95031     AMIT KUMAR PANDEY         ENG433 S   MME210 R

                MME   GN

    3 95067     ASHISH UJWAL               ART402     CS422   S   CS499
CS646

                CSE   GN

    4 95110     GAURAV MUDGAL              ESC202 R   ESO102     MTH331
MTH452     MTH599     SOC171

                MTH   GN

    5 95117     HARISH CHANDRA HARIJAN     CHE671

                CHE   SC      SLOW-B

    6 95145     MANDEEP SINGH CHAUHAN      ESO212 S   PE102B

                CSE   SC      SLOW-A

    7 95208     PUJAK ARORA                CE424      CE482      CE492
EEM608

                CE    GN

    8 95265     SANJEEV KUMAR              BSO209 R   MTH211 R   MTH455 R
MTH653

                MTH   GN

    9 95323     VIJAY KUMAR                CHE211 R   CHE453     ENG433 S
ESO104 R
```

The list of the open electives for 1st semester obtained from Dean (Academic Affairs) office:

| | | | |
|---|---|---|---|
| CS633 | CS645 | CS648 | CS660 |
| CS663 | CS699 | CS720 | CS730 |
| CS799 | EE200 | ESC202 | EEM601 |
| IME521 | ME613 | MME424 | MME452 |
| MME456 | MME470 | MME472 | MME480 |
| MME484 | MME485 | MME631 | MME635 |
| MS606 | NT611 | | |

We modified our program to suit this format of the data.

We first ran the simple apriori over the data and obtained the itemsets involving all the courses thus we ended up having 740 rules for a sub part of the data. Most of them were unwanted associations or itemsets. We wanted to generate the frequent itemsets from among all the open electives only. We tried our tid-list based program for this purpose and found it suited to our cause. One can enter courses of one's choice in the file open_elec. This file is read by the program and the itemsets are generated from these courses only. Then the program is run and the support is found for the combinations of the courses. This reduces our search space considerably and limits the list to only the combinations of our choice. By running the program first on the data before Add/Drop and then on the data after Add/Drop and then comparing the results we can get some more insight of the choice of the students and their inclination towards different courses.

**6.3 Results:**

Though we could not get more interesting patterns for this real data because of very low value of support for any combination but still it gives some idea.

Few pages of result which were obtained after running Simple Apriori on registration data of UG students of 98 batch are attached in the appendix (A). The output contains 5 such pages for the frequent itemsets and more pages for the association rules. This output is exhaustive. In this output we have obtained many combinations which are not giving any useful information. Like the support for the combination of compulsory courses which is apriori known to every one. The Apriori is generating the combinations of all the courses (core courses as well as open elective)

and finds the support of all of them. But we are required to find the support for the combination of the open electives only. So we used the vertical database layout (tid-list) based program for the ARM on this data. In appendix (B) we have shown the complete results for the whole UG and PG data generated from tid-list based program. One can analyze the results as follows.

Suppose for the combination of any two open electives we have some support greater than the support threshold then we should not place those two courses in the same slot and in the same class room. That is we can modify our time table by putting additional constraints with the help of Association Rule Mining on the registration data of previous years.

We see that rule MME424 => MME470 has a support of 1.3% but has the confidence of 100% this shows that all the students who opt for MME424 also opt for MME 470. We should therefore place these two courses in the time table in two different slots.

Comparing the results for the data before ADD/DROP and after ADD/DROP we get an insight how because of clashing and other reasons the students drop the courses.

Course combination CS633, CS660 was taken by 8 students before add/ drop but after add/drop this number drops to 5 only. It might be because of clash in the timetable of these two courses. A similar statement may be made for many other combinations also.

# Chapter 7

## Available Data Miners and Their Features

There are many companies offering Data Mining Softwares. Here we present the features of some of them. The matter below presents what they say about their products Due to lack of time and also because all of them are commercial products, we could not attest their statements.

## 1.Oracle Data Mining Suite (Oracle Darwin):

is powerful enterprise data mining software that finds meaningful patterns hidden within corporate and e-business data — patterns that can provide the insight needed for highly personalized and profitable customer relationships. Oracle Data Mining Suite enables 1·1 marketing by segmenting customers and predicting their behavior.

**Features:**

1. **Enterprise Scalability**
   - **Fully Scalable and Very Large-Database Capable**

   Oracle Data Mining imposes no limit on the amount of data to be mined, so you can exploit all available computer and data resources for maximum benefit
   - **Fast Data Mining**

   Oracle Data Mining utilizes parallel implementations of data mining algorithms to tackle the data mining process in parallel, yielding rapid information discovery. By mining more data faster, superior modeling results are delivered in record time
   - **Deployable Business Models**

   With a single click, Oracle Data Mining generates models in C, C++, or Java code for easy integration with customer "touchpoints" across the enterprise — such as call center, campaign management, and Web-based applications.
2. **Comprehensive Modeling**
   - **Fast Data Access**

   Oracle Data Mining's one-click data-import wizards access Oracle databases and data warehouses via a direct interface (OCI) for rapid import and export of data.

- **Multi-Algorithmic Approach**

A comprehensive array of techniques increases modeling accuracy. Oracle Data Mining features parallel implementations of classification and regression trees, neural networks, k-nearest neighbors (memory-based reasoning), regression, and clustering algorithms.

3. **Ease of Use**

- **Intuitive Windows GUI**

Oracle Data Mining combines familiar Windows ease of use with the power of a fully scalable, UNIX server-based solution.

- **Model Seeker, Key Fields, and Modeling Wizards**

Easy-to-use wizards automate the data mining process, while providing expert users with full control over all advanced options.

Provider's Address: www.oracle.com

## 2. Magnum Opus:

**Magnum Opus** finds association rules from data. **Magnum Opus** can handle large data sets containing up to millions of records, although its capacity to do so will be constrained by the amount of memory available. It empowers the user to select between five measures of the importance of an association, leverage, lift, strength, coverage, or support, providing alternative analyses of the same data.

**Magnum Opus** allows the user to specify the maximum number of association rules to be found, and place restrictions on association rules to be considered. Within these restrictions, it finds the associations with the highest values on the specified measure of importance. It will only find fewer than the specified number of association rules if the search is terminated by the user or there are fewer than the specified number of associations that satisfy the user specified constraints.

Some other important features:

- **Magnum Opus** can automatically filter out associations that are unlikely to be of interest.
- **Magnum Opus** finds association rules from both basket data and attribute-value data.

- **Magnum Opus** has been designed to analyse **substantial databases** containing up to millions of records (although its capacity to do so may be limited by the amount of memory installed in the host computer).
- Unlike other association rule discovery systems, **Magnum Opus** does not rely on sparse data for **efficient processing.**
- **Magnum Opus** is **fast.** It has **linear compute time.** If the amount of data is doubled then the compute time will approximately

Provider's Address: http://www.rulequest.com/MagnumOpus-info html

## 3. Wiz Rule

WizRule is an innovative data auditing and cleansing application that automatically reveals all the rules in a given data, and points at the deviations from the set of the discovered rules as suspected errors.

**Key Features:**

- Reveals *all* the if-then rules,
- Reveals the mathematical formula rules,
- Reveals spelling errors in names and values,
- Calculates the level of unlikelihood of each deviation,
- Avoids false alarms - almost every deviation having a high level of unlikelihood is indeed an error or at least a case to be examined.

WizRule works with any PC-based database either directly or via ODBC connections

WizRule analyses one flat file or joins several files to one table

- Addressing the full data mining process.
- A feature rich package for both the experienced data mining practitioner and for the first time user.
- ActiveX data mining components for embedding within vertical applications
- Scalable high performance client-server data mining.

Provider's Address. **http://www.wizsoft.com**

## 4. Super query

**Description**

SuperQuery is a database query and analysis tool. It automatically displays graphs and statistics about the data. SuperQuery also discovers all the patterns in your data by

Management system, which could have been built in any environment (VB, Delphi, etc.). All this is achieved without compromising scalability or performance.

With this flexible deployment, applications can be delivered as stand alone, embedded within another software environment or as Intranet/Internet enabled applications.

Reporting facilities within Miner are extensive. Visualisation and exploration of the data can be incorporated at any stage of the data mining life cycle.

Data can be filtered through tree profiles and reported on, or extracted to new data sources. Graphs, charts and tables can be dynamically linked to the graphical tree views of your data. As the user navigates through different levels and applies different scenarios they can be kept constantly updated.

XpertRule Miner provides both the feature rich environment required by the experienced data mining practitioner, as well as enabling highly focussed mining and reporting systems to be provided for users who are new to data mining.

## 7. XAffinity™

It was designed to provide the utmost flexibility and power in association and sequence pattern analysis. Whether retail or e-tail, *XAffinity*™ can help find affinities that can markedly increase one's bottom line. *XAffinity*™ provides exceptional flexibility for association and sequence pattern detection and analysis.

**Features:**

- **Standalone and ActiveX DLL Availability**

  *Stand-alone program. XAffinity*™ is available as a standalone program for interactive use by marketers and business analysts.

  *ActiveX DLL.* As an ActiveX DLL, *XAffinity*™ can be embedded in your application programs. All of the features of the standalone product – and more - are available through the DLL, allowing complete integration into software which meets your unique requirements.

- **Database Integration.** *XAffinity*™ does *DBMS embedded* mining. This means that mining is done directly from tables or views in one's database and mining results are placed back into the database.

- ***Direct data access.*** Mining is done directly on transaction data, eliminating the need to reformat or pre-aggregate your data.

- ***Automatic partitioning.*** Partitioning lets you automatically generate rules by any other attribute - store, time of day or day of week - or - for easy comparison.
- ***Rule Discriminator.*** The Discriminator compares rules across user specified analyses or partitions to help you locate rules with the greatest (or least) variability, e.g. across stores, time periods or type of customer.
- ***Hierarchy support.*** Mining can be done at the product or web page, category or department level, or any level in between. You have complete flexibility over how you group items.
- ***Selective rule generation.*** Rules can be selectively generated for specific items on either or both the right hand side and/or the left hand side.
- ***Multiple rule formats.*** Rules are produced in tabular or natural language formats. The natural language format makes rules easy to read and understand.

Provider's Address for 5 and 6: www.attar.com

## 8. Data Mining Products from IBM

***IBM DB2 Intelligent Miner for Data*** provides a single framework to support the iterative, exploratory process of data mining. Analysts prepare, analyze and mine enterprise data to discover unknown patterns and associations, and to develop predictive models based on past performance. Proven, scaleable mining algorithms are used for applications such as customer segmentation, store profiling, attrition analysis, fraud and abuse detection, market basket analysis, cross selling, risk assessment, causal analysis and best practices.

***IBM DB2 Intelligent Miner Scoring*** is an economical and easy-to-use mining deployment capability. It enables users to incorporate mining analytics into BI, eCommerce and OLTP applications. Applications score records (segment, classify or rank the subject of those records) based on a set of predetermined criteria expressed in a data mining model. These applications can better serve business and consumer users alike - to provide more informed recommendations, to alter a process based on past behavior, to build more efficiencies into the online experience; to, in general, be more responsive to the specific situation at hand All scoring functions offered by the Intelligent Miner for Data are supported.

Provider's Address: http://www-4.ibm.com/software/data/iminer/fordata/

# Chapter 8

## Conclusion

Association Rule Mining in Market basket Data is still a challenging task. The main concern is the efficiency of the Algorithms i.e. the time take by an algorithm for ARM. We implemented different existing algorithms for association rule mining and studied their performances. We also took a view into how the performance of a module for ARM is affected adversely by number of partitions of the data, which is done to solve the main memory constraint.

We next proposed a method of local pruning for the distributed data mining that may help increase the efficiency of the count distribution.

We have proposed a new approach for the Association Rule Mining and shown its comparative performance with Apriori Algorithm for the same database and found that our approach is competing well with Apriori. The simplification of the data structure used and the compression of the database are two main features of this algorithm.

The application of the ARM techniques to real world data of registration of IITK students and the results thus obtained are also shown. One can use these results for the purpose of time tabling as they represent additional constraints based on previous years' data.

# Bibliography


[1] R Agrawal, "Database Mining: A Performance Perspective", IEEE Transactions on Knowledge and Data Engineering, Vol. 5, No. 6, December 1993, pp- 914-925.

[2] Bhavani Thuraisingham, "A Primer for Understanding and Applying Data Mining", IT Pro January | February 2000, pp 28-31.

[3] R. Agrawal, Ramakrishnan Srikant, " Fast Algorithm for Mining Association Rules", Proceedings of the 20th VLDB Conference Santiago, Chile, 1994, pp 487- 499.

[4] Ming-Syan Chen, Jiawei Han and Philip S. Yu, " Data Mining: An Overview from a Database Perspective", IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 6, December 1996, pp 866-883.

[5] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman and Shalom Tsur, " Dynamic Itemset Counting and Implication Rules for Market Basket Data", Sigmod records (ACM Special Interest Group on Management of Data), 26(2) 255, 1997.

[6] R. Agrawal, John C. Shafer, " Parallel Mining of Association Rules", IEEE transactions on Knowledge and Data Engineering, Vol. 8, No. 6, December 1996, pp 962-969.

[7] Mohammed J. Zaki, " Scalable Algorithm for Association Mining", IEEE Transactions on Knowledge and Data Engineering", Vol. 12, No 3, May/June 2000, pp 372-390.

[8] Eui- Hong (Sam) Han, George Karypis and Vipin Kumar, " Scalable Parallel Data Mining for Association Rule Mining", IEEE transactions on Knowledge and Data Engineering", Vol. 12, No. 3, May/June 2000 pp 337-351.

[9] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth and ramasamy Uthurusamy (EDITORS), " *Advances in Knowledge Discovery and Data Mining*", AAAI Press/ the MIT Press Cambridge, Massachusetts, 1996.

[10] Hannu Toivonen, " Sampling Large Databases for Association Rules" Proceedings of the 22nd VLDB Conference Mumbai (Bombay), India, 1996, pp 134-145.

[11] Sergey Brin, Rajeev Motwani and Craig Silverstein, " Beyaond Market Baskets. Generalizing Association Rules to Correlations", Proceedings of ACM SIGMOD, 97 AZ, USA. pp 265-276.

[13] Mohammed J. Jaki, " Parallel and Distributed Association Mining: A Survey" *IEEE Concurrency*, special issue on Parallel Mechanisms for Data Mining, Vol. 7, No. 4, pp14-25, December, 1999.

[14] Pieter Adriaeins, Dolf Zantinge, " *Data Mining*", Pearson Education Asia, Addison Wesley, 2000, pp 47.

[15] Ramez Elmasri, Shamkant B. Navathe, " *Fundamentals of Database Systems*", Addison Wesley, 3ed

[16] http.//www pcc qub.ac uk/tec/courses/datamining/ohp/dm-OHP-final_2.html#HEADING4

[17] Mohammed J. Jaki, " Scalable Algorithms for Association Mining", IEEE transactions on Knowledge and data Engineering, Vol. 12, No. 3, May/June 2000.

# Appendix

## Results obtained by running Apriori on Registration data

For 98 UG batch. Autumn Semester

The frequent itemsets are generated at : MinSup:5.000%, MinConf:50.
LevelLimit : 4

==Number of Transactions == 367

Item 1:
ECO414 = Y
(10.354% 38)

Item 2:
IME521 = Y
(8.992% 33)

Item 3:
SOC470 = Y
(7.357% 27)

Item 4:
TA202 = Y
(7.902% 29)

Item 5:
SOC486 = Y
(7.902% 29)

Item 6:
NT611 = Y
(8.447% 31)

Item 7:
PSY468 = Y
(8.447% 31)

Item 8:
ENG434 = Y
(8.992% 33)

Item 9:
ENG431 = Y
(7.902% 29)

Item 10:
CE273 = Y
(10.899% 40)

Item 11:
CE311 = Y
(10.899% 40)

Item 12:
CE321 = Y
(11.172% 41)

Item 13:
CE331 = Y
(10.899% 40)

Item 14:
PHI447 = Y
(6.267% 23)

Item 15:
ECO412 = Y
(6.812% 25)

Item 16:
PSY458 = Y
(9.809% 36)

Item 17:
BSO203 = Y
(18.529% 68)

Item 18:
CHE312 = Y
(12.262% 45)

Item 19:
CHE313 = Y
(12.262% 45)

Item 20:
CHE331 = Y
(12.262% 45)

Item 21:
ESO210 = Y
(19.619% 72)

Item 22:
CS301 = Y
(11.717% 43)

Item 23:
CS330 = Y
(11.172% 41)

Item 24:
CS350 = Y
(11.172% 41)

Item 25:
CS355 = Y
(11.172% 41)

Item 26.
CS360 = Y
(6.540% 24)

Item 27:
EE320 = Y
(16.894% 62)

Item 28:
EE330 = Y
(18.529% 68)

Item 29:
EE370 = Y
(17.711% 65)

Item 30:
EE380 = Y
(17.166% 63)

Item 31:
ME331 = Y
(16.076% 59)

Item 32:
ME351 = Y
(16.076% 59)

Item 33:
ME353 = Y
(16.076% 59)

Item 34:
ME371 = Y
(16.076% 59)

Item 35:
MME310 = Y
(10.899% 40)

Item 36:
MME320 = Y
(11.172% 41)

Item 37:
MME331 = Y
(10.627% 39)

Item 38:
MME340 = Y
(11.172% 41)

Item 39:
CE311 = Y
CE273 = Y
(10.899% 40)

Item 40:
CE321 = Y
CE273 = Y
(10.899% 40)

Item 41:
CE331 = Y
CE273 = Y
(10.899% 40)

Item 42:
CE321 = Y
CE311 = Y
(10.899% 40)

Item 43:
CE331 = Y
CE311 = Y
(10.899% 40)

Item 44:
CE331 = Y
CE321 = Y
(10.899% 40)

Item 45:
EE320 = Y
BSO203 = Y
(16.894% 62)

Item 46:
EE330 = Y
BSO203 = Y
(17.166% 63)

Item 47:
EE370 = Y
BSO203 = Y
(17.166% 63)

Item 48:
EE380 = Y
BSO203 = Y
(16.894% 62)

Item 49:
CHE313 = Y
CHE312 = Y
(12.262% 45)

Item 50:
CHE331 = Y
CHE312 = Y
(12.262% 45)

Item 51:
CHE331 = Y
CHE313 = Y
(12.262% 45)

Item 52:
ME331 = Y
ESO210 = Y
(15.804% 58)

Item 53:
ME351 = Y
ESO210 = Y
(15.804% 58)

Item 54:
ME353 = Y
ESO210 = Y
(15.804% 58)

Item 55:
ME371 = Y
ESO210 = Y
(15.804% 58)

Item 56:
CS330 = Y
CS301 = Y
(10.627% 39)

Item 57:
CS350 = Y
CS301 = Y
(10.627% 39)

Item 58:
CS355 = Y
CS301 = Y
(10.627% 39)

Item 59:
CS360 = Y
CS301 = Y
(6.267% 23)

Item 60:
CS350 = Y
CS330 = Y
(11.172% 41)

Item 61:
CS355 = Y
CS330 = Y
(11.172% 41)

Item 62:
CS360 = Y
CS330 = Y
(6.267% 23)

Item 63:
CS355 = Y
CS350 = Y
(11.172% 41)

Item 64:
CS360 = Y
CS350 = Y
(6.267% 23)

Item 65:
CS360 = Y
CS355 = Y
(6.267% 23)

Item 66:
EE330 = Y
EE320 = Y
(16.894% 62)

Item 67:
EE370 = Y
EE320 = Y
(16.894% 62)

Item 68:
EE380 = Y
EE320 = Y
(16.894% 62)

Item 69:
EE370 = Y
EE330 = Y
(17.711% 65)

Item 70:
EE380 = Y
EE330 = Y
(17.166% 63)

Item 71:
EE380 = Y
EE370 = Y

(17.166%
63)

Item 72:
ME351 = Y
ME331 = Y
(16.076%
59)

Item 73.
ME353 = Y
ME331 = Y
(16.076%
59)

Item 74:
ME371 = Y
ME331 = Y
(16.076% 59)

Item 75:
ME353 = Y
ME351 = Y
(16 076% 59)

Item 76:
ME371 = Y
ME351 = Y
(16.076% 59)

Item 77:
ME371 = Y
ME353 = Y
(16.076% 59)

Item 78:
MME320 = Y
MME310 = Y
(10.899% 40)

Item 79:
MME331 = Y
MME310 = Y
(10.354% 38)

Item 80:
MME340 = Y
MME310 = Y
(10.899% 40)

Item 81:
MME331 = Y
MME320 = Y
(10.627% 39)

Item 82:
MME340 = Y
MME320 = Y
(11.172% 41)

Item 83:
MME340 = Y
MME331 = Y
(10.627% 39)

Item 84:
CE321 = Y
CE311 = Y
CE273 = Y
(10.899% 40)

Item 85:
CE331 = Y
CE311 = Y
CE273 = Y
(10.899% 40)

Item 86:
CE331 = Y
CE321 = Y
CE273 = Y
(10.899% 40)

Item 87:
CE331 = Y
CE321 = Y
CE311 = Y
(10.899% 40)

Item 88:
EE330 = Y
EE320 = Y
BSO203 = Y
(16.894% 62)

Item 89:
EE370 = Y
EE320 = Y
BSO203 = Y
(16.894% 62)

Item 90:
EE380 = Y
EE320 = Y
BSO203 = Y
(16.894% 62)

Item 91:
EE370 = Y
EE330 = Y
BSO203 = Y
(17.166% 63)

Item 92:
EE380 = Y
EE330 = Y
BSO203 = Y
(16.894% 62)

Item 93:
EE380 = Y
EE370 = Y
BSO203 = Y
(16.894% 62)

Item 94:
CHE331 = Y
CHE313 = Y
CHE312 = Y
(12.262% 45)

Item 95:
ME351 = Y
ME331 = Y
ESO210 = Y
(15 804% 58)

Item 96:
ME353 = Y
ME331 = Y
ESO210 = Y
(15.804% 58)

Item 97:
ME371 = Y
ME331 = Y
ESO210 = Y
(15.804%
58)

Item 98:
ME353 = Y
ME351 = Y
ESO210 = Y
(15.804%
58)

Item 99:
ME371 = Y
ME351 = Y
ESO210 = Y
(15.804%
58)

Item 100:
ME371 = Y
ME353 = Y
ESO210 = Y
(15.804%
58)

Item 101:
CS350 = Y
CS330 = Y
CS301 = Y
(10.627%
39)

Item 102:
CS355 = Y

CS330 = Y
CS301 = Y
(10.627% 39)

Item 103:
CS360 = Y
CS330 = Y
CS301 = Y
(5.995% 22)

Item 104:
CS355 = Y
CS350 = Y
CS301 = Y
(10.627% 39)

Item 105:
CS360 = Y
CS350 = Y
CS301 = Y
(5.995% 22)

Item 106:
CS360 = Y
CS355 = Y
CS301 = Y
(5.995% 22)

Item 107:
CS355 = Y
CS350 = Y
CS330 = Y
(11.172% 41)

Item 108:
CS360 = Y
CS350 = Y
CS330 = Y
(6.267% 23)

Item 109:
CS360 = Y
CS355 = Y
CS330 = Y
(6 267% 23)

Item 110:
CS360 = Y
CS355 = Y
CS350 = Y
(6.267% 23)

Item 111:
EE370 = Y
EE330 = Y
EE320 = Y
(16.894% 62)

Item 112:
EE380 = Y
EE330 = Y
EE320 = Y
(16.894% 62)

Item 113:
EE380 = Y
EE370 = Y
EE320 = Y
(16.894% 62)

Item 114:
EE380 = Y
EE370 = Y
EE330 = Y
(17.166% 63)

Item 115:
ME353 = Y
ME351 = Y
ME331 = Y
(16.076% 59)

Item 116:
ME371 = Y
ME351 = Y
ME331 = Y
(16.076% 59)

Item 117:
ME371 = Y
ME353 = Y
ME331 = Y
(16.076% 59)

Item 118:
ME371 = Y
ME353 = Y
ME351 = Y
(16 076% 59)

Item 119:
MME331 = Y
MME320 = Y
MME310 = Y
(10.354% 38)

Item 120:
MME340 = Y
MME320 = Y
MME310 = Y
(10.899% 40)

Item 121:
MME340 = Y
MME331 = Y
MME310 = Y
(10.354% 38)

Item 122:
MME340 = Y
MME331 = Y
MME320 = Y
(10.627% 39)

Item 123:
CE331 = Y
CE321 = Y
CE311 = Y
CE273 = Y
(10.899% 40)

Item 124:
EE370 = Y
EE330 = Y
EE320 = Y
BSO203 = Y
(16.894% 62)

Item 125·
EE380 = Y
EE330 = Y
EE320 = Y
BSO203 = Y
(16.894% 62)

Item 126:
EE380 = Y
EE370 = Y
EE320 = Y
BSO203 = Y
(16.894% 62)

Item 127:
EE380 = Y
EE370 = Y
EE330 = Y
BSO203 = Y
(16.894% 62)

Item 128:
ME353 = Y
ME351 = Y
ME331 = Y
ESO210 = Y
(15.804% 58)

Item 129:
ME371 = Y
ME351 = Y
ME331 = Y

ESO210 = Y
(15.804% 58)

Item 130:
ME371 = Y
ME353 = Y
ME331 = Y
ESO210 = Y
(15.804% 58)

Item 131:
ME371 = Y
ME353 = Y
ME351 = Y
ESO210 = Y
(15 804% 58)

Item 132:
CS355 = Y
CS350 = Y
CS330 = Y
CS301 = Y
(10.627% 39)

Item 133:
CS360 = Y
CS350 = Y
CS330 = Y
CS301 = Y
(5.995% 22)

Item 134:
CS360 = Y
CS355 = Y
CS330 = Y
CS301 = Y
(5.995% 22)

Item 135:
CS360 = Y
CS355 = Y
CS350 = Y
CS301 = Y
(5.995% 22)

Item 136:
CS360 = Y
CS355 = Y
CS350 = Y
CS330 = Y
(6.267% 23)

Item 137:
EE380 = Y
EE370 = Y
EE330 = Y
EE320 = Y
(16.894% 62)

Item 138:
ME371 = Y
ME353 = Y
ME351 = Y
ME331 = Y
(16.076% 59)

Item 139:
MME340 = Y
MME331 = Y
MME320 = Y
MME310 = Y
(10.354% 38)

**Association Rules Generated (Sample Pages)**

```
Rule 1:
        CE311 = Y
        ->    CE273 = Y
        (10.899% 100.00% 40 40 10.899%)
Rule 2:
        CE273 = Y
        ->    CE311 = Y
        (10.899% 100.00% 40 40 10.899%)
Rule 3:
        CE321 = Y
        CE311 = Y
        ->    CE273 = Y
        (10.899% 100.00% 40 40 10.899%)
Rule 4:
        CE321 = Y
        CE273 = Y
        ->    CE311 = Y
        (10.899% 100.00% 40 40 10.899%)
Rule 5:
        CE311 = Y
        CE273 = Y
        ->    CE321 = Y
        (10.899% 100.00% 40 40 10.899%)
Rule 6.
        CE321 = Y
        ->    CE311 = Y
              CE273 = Y
        ( 11.172% 97.56% 41 40 10.90%)
Rule 7:
        CE311 = Y
        ->    CE321 = Y
              CE273 = Y
        ( 10.899% 100.00% 40 40 10.90%)
Rule 8:
        CE273 = Y
        ->    CE321 = Y
              CE311 = Y
        ( 10.899% 100.00% 40 40 10.90%)
Rule 9:
        CE331 = Y
        CE321 = Y
        CE311 = Y
        ->    CE273 = Y
        (10.899% 100.00% 40 40 10.899%)
Rule 10:
        CE331 = Y
        CE321 = Y
        CE273 = Y
        ->    CE311 = Y
        (10.899% 100.00% 40 40 10.899%)
Rule 11:
        CE331 = Y
        CE311 = Y
        CE273 = Y
        ->    CE321 = Y
        (10.899% 100.00% 40 40 10.899%)
Rule 12:
        CE321 = Y
        CE311 = Y
```

```
        CE273 = Y
        ->    CE331 = Y
 Rule 527.
        MME310 = Y
        ->    MME340 = Y
        (10.899% 100.00% 40 40 10.899%)
 Rule 528:
        MME331 = Y
        ->    MME320 = Y
        (10.627% 100.00% 39 39 10.627%)
 Rule 529:
        MME320 = Y
        ->    MME331 = Y
        (11.172% 95.12% 41 39 10.627%)
 Rule 530:
        MME340 = Y
        MME331 = Y
        ->    MME320 = Y
        (10.627% 100.00% 39 39 10.627%)
 Rule 531:
        MME340 = Y
        MME320 = Y
        ->    MME331 = Y
        (11.172% 95.12% 41 39 10.627%)
 Rule 532:
        MME331 = Y
        MME320 = Y
        ->    MME340 = Y
        (10.627% 100.00% 39 39 10.627%)
 Rule 533:
        MME340 = Y
        ->    MME331 = Y
              MME320 = Y
        ( 11.172% 95.12% 41 39 10.63%)
 Rule 534:
        MME331 = Y
        ->    MME340 = Y
              MME320 = Y
        ( 10.627% 100.00% 39 39 10.63%)
 Rule 535:
        MME320 = Y
        ->    MME340 = Y
              MME331 = Y
        ( 11.172% 95.12% 41 39 10.63%)
 Rule 536:
        MME340 = Y
        ->    MME320 = Y
        (11.172% 100.00% 41 41 11.172%)
 Rule 537:
        MME320 = Y
        ->    MME340 = Y
        (11.172% 100.00% 41 41 11.172%)
 Rule 538:
        MME340 = Y
        ->    MME331 = Y
        (11.172% 95.12% 41 39 10.627%)
 Rule 539:
        MME331 = Y
        ->    MME340 = Y
        (10.627% 100.00% 39 39 10.627%)
```

Output of Registration data of First semester 2000

(Before ADD/DROP ) After Running tid-list based program

| | | |
|---|---|---|
| CS633 | 13 | 0.011535 |
| CS633 CS645 | 5 | 0.00443656 |
| CS633 CS645 CS660 | 4 | 0.00354925 |
| CS633 CS645 CS660 CS663 | 4 | 0.00354925 |
| CS633 CS645 CS663 | 5 | 0.00443656 |
| CS633 CS648 | 6 | 0.00532387 |
| CS633 CS648 CS663 | 5 | 0.00443656 |
| CS633 CS660 | 8 | 0.00709849 |
| CS633 CS660 CS663 | 7 | 0.00621118 |
| CS633 CS663 | 12 | 0.0106477 |
| CS645 | 19 | 0.0168589 |
| CS645 CS648 | 7 | 0.00621118 |
| CS645 CS648 CS660 | 4 | 0.00354925 |
| CS645 CS648 CS660 CS663 | 4 | 0.00354925 |
| CS645 CS660 | 12 | 0.0106477 |
| CS645 CS660 CS663 | 11 | 0.00976043 |
| CS645 CS663 | 17 | 0.0150843 |
| CS648 | 21 | 0.0186335 |
| CS648 CS663 | 11 | 0.00976043 |
| CS660 | 30 | 0.0266193 |
| CS663 | 37 | 0.0328305 |
| EE200 | 81 | 0.0718722 |
| EE200 ESC202 | 71 | 0.0629991 |
| ESC202 | 186 | 0.16504 |
| IME521 | 45 | 0.039929 |
| IME521 MME470 | 4 | 0.00354925 |
| ME613 | 9 | 0.0079858 |
| MME424 | 7 | 0.00621118 |
| MME424 MME470 | 7 | 0.00621118 |
| MME470 | 8 | 0.0337178 |
| MME631 | 10 | 0.00887311 |
| MME635 | 3 | 0.00266193 |
| NT611 | 47 | 0.0417036 |

THE Time TAKEN BY THE PROGRAM IS 115

**Output of Regıstration data of First semester 2000**

**(After ADD/DROP ) After Running tıd-list based program**

| | | | | |
|---|---|---|---|---|
| CS633 | | | 8 | 0.00709849 |
| CS633 | CS660 | | 5 | 0.00443656 |
| CS648 | | | 21 | 0.0186335 |
| CS648 | CS660 | | 5 | 0.00443656 |
| CS660 | | | 27 | 0.0239574 |
| EE200 | | | 99 | 0.0878438 |
| EE200 | ESC202 | | 87 | 0.0771961 |
| ESC202 | | | 197 | 0.1748 |
| EEM601 | | | 4 | 0.00354925 |
| IME521 | | | 40 | 0.0354925 |
| IME521 | MME470 | | 4 | 0.00354925 |
| IME521 | NT611 | | 3 | 0.00266193 |
| ME613 | | | 16 | 0.014197 |
| ME613 | NT611 | | 5 | 0.00443656 |
| MME424 | | | 14 | 0.0124224 |
| MME424 | MME470 | | 14 | 0.0124224 |
| MME424 | MME470 | NT611 | 3 | 0.00266193 |
| MME424 | NT611 | | 3 | 0.00266193 |
| MME470 | | | 41 | 0.0363798 |
| MME470 | MME631 | | 17 | 0.0150843 |
| MME470 | MME631 | NT611 | 6 | 0.00532387 |
| MME470 | MME635 | | 3 | 0.00266193 |
| MME470 | NT611 | | 9 | 0.0079858 |
| MME631 | | | 18 | 0.0159716 |
| MME631 | NT611 | | 6 | 0.00532387 |
| MME635 | | | 3 | 0.00266193 |
| NT611 | | | 63 | 0.0559006 |

THE Time TAKEN BY THE PROGRAM IS 43

**Output for the data of second semester 2001**

| | | | |
|---|---|---|---|
| CS425 | | 66 | 0.0417193 |
| CS425 | CS646 | 4 | 0.00252845 |
| CS646 | | 11 | 0.00695322 |
| CS673 | | 6 | 0.00379267 |
| EE301 | | 81 | 0.051201 |
| EE301 | EE360 | 76 | 0.0480405 |
| EE360 | | 78 | 0.0493047 |
| ENG444 | | 17 | 0.0107459 |
| IME624 | | 30 | 0.0189633 |
| IME624 | IME681 | 4 | 0.00252845 |
| IME647 | | 6 | 0.00379267 |
| IME681 | | 18 | 0.011378 |
| MME467 | | 24 | 0.0151707 |
| MME467 | MME609 | 4 | 0.00252845 |
| MME467 | MS612 | 5 | 0.00316056 |
| MME471 | | 18 | 0.011378 |
| MME471 | MME613 | 6 | 0.00379267 |
| MME471 | MME619 | 5 | 0.00316056 |
| MME609 | | 8 | 0.00505689 |
| MME613 | | 36 | 0.022756 |
| MME619 | | 22 | 0.0139064 |
| MME619 | MS612 | 5 | 0.00316056 |
| MS612 | | 9 | 0.005689 |
| NT641 | | 19 | 0.0120101 |
| PHY306 | | 16 | 0.0101138 |

THE Time TAKEN BY THE PROGRAM IS 87

# INDIAN INSTITUTE OF TECHNOLOGY KANPUR
# POST GRADUATE OFFICE

No A/PG/M Thesis/ Dated 26th MARCH 2001

To,

| | |
|---|---|
| Dr T V Prabhakar | CSE |
| Dr R R K Sharma | IME |
| Dr Veena Bansal | IME |

Please find herewith an unbound copy of the thesis submitted by the following student in partial fulfillment of the requirements of **M.Tech** degree **in INDUSTRIAL MANAGEMENT ENGINEERING** / Science of the Institute The date and time of his **ORAL EXAMINATION** will be intimated to you direct by the department concerned

| **ROLL No.** | **NAME OF STUDENT** | **THESIS TITLE** |
|---|---|---|
| 9911411 | PUNIT KUMAR MISHRA | EFFICIENT MINING OF ASSOCIATION RULES IN LARGE DATABASE |

Krishna Kumar

Dean
Academic Affairs

Encl As above

C C to
The Convener, DPGC, Department of IME